ATS962x Coprocessor
FPGA Development Kit (FDK)

User's Guide

Version 1.0.2
November 9, 2012

License Agreement

Important
By using this software and FPGA design you accept the following terms of this License Agreement. If you do not agree with these terms, you should not use the software and FPGA design promptly return them for a refund.

Ownership
Alazar Technologies, Inc., retains the ownership of this copy of the enclosed software and FPGA design package. It is licensed to you for use under the following conditions:

Grant of License
You may only concurrently use the enclosed software and/or FPGA design on the computers that have an Alazar Technologies, Inc. waveform digitizer card plugged in (for example, if you have purchased one Alazar Technologies, Inc. card, you have a license for one concurrent usage). If the number of users of the software and/or FPGA design exceeds the number of Alazar Technologies, Inc. cards you have purchased, you must have a reasonable process in place to assure that the number of persons concurrently using the software and/or FPGA design does not exceed the number of Alazar Technologies, Inc. cards purchased.

You may transfer this software and FPGA design to another party if the other party agrees to the terms and conditions of the agreement and completes and returns a registration card to Alazar Technologies, Inc. The registration card is available by writing to Alazar Technologies, Inc. If you transfer the software and FPGA design, you must simultaneously transfer all documentation and related disks.

Restrictions
You may not copy the documentation or software and/or FPGA design except as described in the installation section of this manual. You may not distribute, rent, sub-lease or lease the software and/or FPGA design or documentation, including translating, decomposing, or disassembling, or creating derivative works. You may not reverse-engineer any part of this software and/or FPGA design, or produce any derivative work. You may not make telecommunication transmittal of this software and/or FPGA design.

Termination
This license and your right to use this software and FPGA design automatically terminates if you fail to comply with any provision of this license agreement.

Rights
Alazar Technologies, Inc. retains all rights not expressly granted. Nothing in this agreement constitutes a waiver of Alazar Technologies, Inc.'s rights under the Canadian and U.S. copyright laws or any other Federal or State law.

© 2003-2012 Alazar Technologies Inc.

Limited Warranty

If you discover physical defects in the media, Alazar Technologies, Inc. will replace the media or documentation at no charge to you, provided you return the item to be replaced with proof of payment to Alazar Technologies, Inc. during the 90-day period after having taken delivery of the software and FPGA design.

Alazar Technologies, Inc. excludes any and all implied warranties, including warranties of merchantability and fitness for a particular purpose and limits your remedy to return the software and FPGA design and documentation to Alazar Technologies, Inc. for replacement. Although Alazar Technologies, Inc. has tested the software and FPGA design and reviewed the documentation, ALAZAR TECHNOLOGIES, INC. MAKES NO WARRANTY OF REPRESENTATION, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THIS SOFTWARE AND FPGA DESIGN OR DOCUMENTATION, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE AND FPGA DESIGN AND DOCUMENTATION IS LICENSED “as is” AND YOU, THE LICENSEE, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE. IN NO EVENT WILL ALAZAR TECHNOLOGIES, INC. BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THIS SOFTWARE AND FPGA DESIGN OR DOCUMENTATION, even if advised of the possibility of such damages. In particular, Alazar Technologies, Inc. shall have no liability for any data acquired, stored or processed with this software and/or FPGA design, including the costs of recovering such data.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESSED OR IMPLIED. No Alazar Technologies, Inc. dealer, agent or employee is authorized to make any modifications or additions to this warranty.

Information in this document is subject to change without notice and does not represent a commitment on the part of Alazar Technologies, Inc. The software and FPGA design described in this document is furnished under this license agreement. The software and/or FPGA design may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the software and/or FPGA design on any medium except as specifically allowed in the license agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the written permission of Alazar Technologies, Inc.

Some jurisdictions do not allow the exclusion of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights, which vary from jurisdiction to jurisdiction.

© 2003-2012 Alazar Technologies Inc.

Contents

PREAMBLE

1 FDK Overview v
1.1 How to use the Books in this Guide v
1.2 About the Frameworks and Example Designs v

BOOK I - FDK Designer's Guide (SingleClk Framework)

2 Book I - Introduction 7
3 Quick Start Guide 13
4 Design Description 24
5 Design Recommendations 44
6 Software and System Considerations 50
7 FPGA Compilation Environment 55
8 Board-Level Debugging Facilities 57
9 Simulation Framework 62

BOOK II - FDK SingleClk Framework with FIR & DDC Example

10 Book II - Introduction (SingleClk FIR & DDC) 79
11 Functional Description 80
12 Detailed Functional Description 84
13 Memory Map 90
14 Special Considerations 103
15 Quick Start Guide 104

BOOK III - FDK MultiClk Framework with FFT Example

16 Book III - Introduction (MultiClk & FFT) 114
17 Functional Description 118
18 Detailed Functional Description 128
19 Memory Map 144
20 Miscellaneous Considerations 155
21 Software and System Considerations 156
22 FPGA Designer Section 157
23 Design Recommendations 161
24 Simulation Framework 162
25 Glossary 163

© 2003-2012 Alazar Technologies Inc.

1 FDK Overview

1.1 How to use the Books in this Guide

This single document in structured in 2 Books:

- BOOK I - FDK Designer's Guide
- BOOK II - FDK SingleClk Framework with FIR & DDC Example
- BOOK III - FDK MultiClk Framework with FFT Example

Book I covers all the bases of the FPGA Development Kit and should be read by everyone wishing to make use the FDK and make customizations or FPGA development. The complete FDK development framework is described, along with design considerations, a QuickStart guide, etc.
In the base framework, a single clock (SingleClk) domain is used for the datapath.

Book II builds upon Book I by adding a DSP design example to the framework. This example provides FIR filter functionality as well as a Digital Down-Converter functionality.

Book II also builds upon Book I in 2 ways:

1. MultiClk framework: It describes a more sophisticated framework whereby multiple clock domains are used in the datapath, in order to provide more design flexibility.
2. FFT example: A fully-functional FFT sample design has been implemented by AlazarTech within the MultiClk framework. This design illustrates a number of advanced features.

Each book has its own Table of Contents.

1.2 About the Frameworks and Example Designs

The Single-clock (SingleClk) framework represents the base framework for the entire FDK. It consists of a complete design, simulation and build environment along with VHDL source code in which the user can add their custom functionality. The Analog-to-Digital Converters' clock is used throughout the entire datapath, hence the SingleClk name. This is the simplest framework.

The base framework transfers data Transparently (no signal processing) through the Coprocessor FPGA.

The FIR & DDC example is a direct addition to the SingleClk framework and serves useful functionality while demonstrating how to add practical functionality to the framework.

© 2003-2012 Alazar Technologies Inc.

The MultiClk framework modifies the SingleClk framework, and introduces Multiple clock domains for the datapath. This adds flexibility but also adds some complexity, in terms of flow-control in the datapath and other considerations.

The MultiClk framework comes with an example design which implements a Fast-Fourier Transform (FFT), as well as other datapath options. It is worth noting that the MultiClk framework and the FFT example are provided as one single unit of source code.

© 2003-2012 Alazar Technologies Inc.

Book I - FDK Designer's Guide

-- BOOK I --

FDK Designer's Guide
(and SingleClk Framework)

© 2003-2012 Alazar Technologies Inc.

Book I Contents

2 Book I - Introduction 7
2.1 Overview of the FPGA Development Kit 7
2.2 Software Requirements 7
2.3 Quick Start Guide 7
2.4 What is provided in the FPGA Development Kit 7
2.5 Hardware Overview 8
2.5.1 Coprocessor FPGA (CPF) Device Information 8
2.6 Development Framework Overview 9
2.6.1 Coprocessor FPGA Design Framework 9
2.6.2 Sample User Design 9
2.6.3 Quartus FPGA project 10
2.6.4 Simulation Framework and Testbench 10
2.6.5 Development Environment Facilitators 10
2.7 Pre-requisites 10
2.7.1 FPGA Design Expertise 11
2.7.2 Tools, Licenses and Equipment 11
2.7.3 Design Responsibility 12
2.8 Contacting us 12
3 Quick Start Guide 13
3.1 Installing the ATS962x Coprocessor FDK 13
3.2 Implementing your Design 13
3.2.1 Where to insert your design 13
3.3 Simulating the Design 13
3.4 Compiling the Design and Producing a Downloadable File 16
3.5 Downloading the FPGA to the Board 18
3.5.1 FPGA Download using AlazarTech DSO Software 18
3.5.2 FPGA Download using JTAG and Signal Tap II 20
3.5.3 FPGA Download through software (ATS-SDK) 21
3.6 Testing the User Design on a board 22
3.7 Design Modification Reminders 23
4 Design Description 24
4.1 Architecture 24
4.1.1 ATS9625 Waveform Digitizer Board 24
4.1.2 Coprocessor FPGA 25
4.1.3 File Folder Hierarchy 25
4.2 Scope of Design Provided 26
4.2.1 User's Responsibility 26
4.3 Port Description - User Interface (user_top_level.vhd) 26
4.4 Datapath Functional Description 28
4.4.1 Datapath/Trigger/Aux Latency Matching Considerations 29
4.4.2 Trigger Pulse Requirements 30
4.4.3 Data Format 30
4.4.4 Out-Of-Range (OOR) 30
4.5 Trigger Signals 31

© 2003-2012 Alazar Technologies Inc.

4.6 Auxiliary Ports 31
4.6.1 Faceplate <-> Coprocessor FPGA 31
4.6.2 Coprocessor FPGA <-> Main FPGA 31
4.6.3 Typical Auxiliary Port Configuration 32
4.7 Control Bus 32
4.7.1 Control Bus Interface Functional Description 32
4.7.2 Memory Map Framework 33
4.7.3 How to Add New Registers 34
4.8 Reset Signals, Startup Sequence, and FPGA Download 35
4.9 Clocks 35
4.9.1 user_adc_clk 35
4.9.2 clk_osc_50mhz 36
4.9.3 ctl_bus_gated_clk 36
4.9.4 Other Clocks (Advanced) 37
4.9.4.1 enc_clk 37
4.9.4.2 c2m_data_clk_b 37
4.10 LEDs and Debug I/O's 37
4.11 Memory Map 37
4.11.1 Memory Map - Alazar Section (Reserved) 39
4.11.1.1 cpf_version Register Description 39
4.11.1.2 cpf_variant Register Description 39
4.11.1.3 compilation_timestamp Register Description 40
4.11.1.4 cpf_config Register Description 40
4.11.1.5 cpf_status Register Description 40
4.11.2 Memory Map - User Section, Alazar-Defined 41
4.11.2.1 user_version Register Description 41
4.11.2.2 data_format_cha/chb Register Description 42
4.11.3 Memory Map - User Section, Open (Customizable) 43
5 Design Recommendations 44
5.1 Where to insert your design 44
5.2 Please do not modify Alazar source files! 44
5.3 Clocking Schemes 44
5.3.1 Single-Clock Scheme (Recommended) 45
5.3.2 Multiple Clock Schemes 46
5.3.2.1 50Mhz Oscillator vs. ADC Sampling Clock Domains 46
5.3.3 PLLs 47
5.3.3.1 A Warning about using PLLs 47
5.4 Memory Initialization Files 47
5.4.1 RAMs and ROMs 47
5.4.2 Complex IP Blocks 48
5.4.3 Scripts copying .hex and .mif files to Project directory 48
5.5 VHDL and Naming Conventions 48
5.5.1 Prefixes 48
5.5.2 Suffixes 49
5.5.3 Block structures 49
6 Software and System Considerations 50

© 2003-2012 Alazar Technologies Inc.

6.1 Acquisition Transfer Modes 50
6.1.1 All Modes other than Coprocessor-Managed Mode 50
6.1.2 Coprocessor-Managed Mode 51
6.1.2.1 Acquisition Control & Data Transfer Mechanism 52
6.1.2.2 End of Acquisition & Buffer Transfer Considerations 53
6.1.2.3 Summary of Recommendations 53
7 FPGA Compilation Environment 55
7.1 Overview 55
7.2 FPGA type - Selecting the High-Capacity (HC) Revision 55
7.3 File Descriptions 55
7.3.1 Quartus Project Files (.qpf, .qsf) 55
7.3.2 FPGA Programming Files (.rbf, .sof) 55
7.3.2.1 coprocessor.rbf 56
7.3.2.2 coprocessor.sof 56
7.3.3 Timing Constraints file (.sdc) 56
7.3.4 Quartus Custom Script files 56
7.3.4.1 custom_tq_script.tcl 56
7.3.4.2 quartus_pre_flow_script.tcl 56
8 Board-Level Debugging Facilities 57
8.1 AlazarDSO User Interface 57
8.2 AlazarTech JTAG Debug Board 58
8.3 Altera SignalTap II® 59
8.4 Altera In-System Memory Content Editor® 60
9 Simulation Framework 62
9.1 Overview 62
9.1.1 Simulation Folder Hierarchy 62
9.2 Facilitator Scripts 63
9.2.1 simbuild.bat script 63
9.2.2 simrun script 65
9.2.3 modelsim.tcl 65
9.3 Wavefile (wave.do) 65
9.4 SlickEdit® Workspace and Project Files 67
9.5 Testbench 67
9.5.1 Features 68
9.5.2 File and Module Descriptions 68
9.5.2.1 Basic Sine Wave Generator 69
9.5.2.2 Arbitrary Waveform Data Generator Module 69
9.5.2.3 Waveform Sample Generator Spreadsheet 69
9.5.2.4 Data capture-to-file Module 70
9.5.3 Testbench Code Description 70
9.5.3.1 Testcase Master Process 70
9.5.3.2 Self-Monitored Tests and Testcase Pass / Fail Reporting 70
9.5.3.3 Control Bus Accesses 72
9.5.3.4 Verbosity Control 73
9.5.3.5 Stimulus Generators 73
9.5.3.6 Sample Analyzer: Trigger Monitor 73

© 2003-2012 Alazar Technologies Inc.

9.5.3.7 c2m_data_channel_separator process 73
9.5.3.8 Optional Expansion to Multiple Testcases 73

© 2003-2012 Alazar Technologies Inc.

List of Figures

Figure 4-1 ATS9625 Board Block Diagram 24
Figure 4-2 Package File Folder Hierarchy 25
Figure 4-3 Functional Timing Diagram - Datapath 29
Figure 4-4 Functional Timing Diagram - Control Bus 33
Figure 4-5 Memory Map Address Constants in user_mem_map_pkg.vhd 34
Figure 8-1 AlazarDSO Coprocessor FPGA Menu 57
Figure 8-2 AlazarDSO Coprocessor Window 58
Figure 8-3 Connecting the JTAG Debug Board 59
Figure 8-4 Signal Tap II® Sample Display 60
Figure 8-5 In-System Memory Content Editor® Sample Screenshot 61
Figure 9-1 simbuild.bat script sample output (abbreviated) 64
Figure 9-2 Wavefile - Datapath sample 66
Figure 9-3 Wavefile - Control Bus sample 67
Figure 9-4 Sample Transcript of all tests passed 71
Figure 9-5 Sample Transcript of a few failing tests 72

List of Tables

Table 2-1 Coprocessor FPGA Resources 9
Table 4-1 User Port Interface Signals 26
Table 4-2 Faceplate to CPF Auxiliary Ports 31
Table 4-3 CPF to Main FPGA Auxiliary Ports 31
Table 4-4 Memory Map - Global 37
Table 4-5 Memory Map - Alazar Reserved Section 39
Table 4-6 Memory Map - User Customizable Section 41
Table 4-7 Memory Map - User Customizable Section 43

© 2003-2012 Alazar Technologies Inc.

2 Book I - Introduction

This document describes the contents of the ATS962x Coprocessor FPGA Development Kit (FDK), as well as how to use it. Although much of the text refers to ATS9625, it also applies to all members of the ATS962x family, such as the ATS9626.

2.1 Overview of the FPGA Development Kit

The FPGA Development Kit consists in a FPGA development framework, including source files, which provides the user with access to very powerful hardware-speed signal processing capabilities. It allows the user to develop custom intellectual property independently.

2.2 Software Requirements

The ATS962x Coprocessor FDK requires the following software versions:

- ATS9625 (or ATS9626) driver v5.9.1 or higher (required for all applications).
- AlazarDSO v1.1.36 or higher (if using AlazarDSO).
- ATS-SDK v6.0.4 (if using an SDK).

The driver is required in all cases, but note that the user has the choice of using an ATS-SDK or AlazarDSO as an interface, or both.

Please note this section lists the Alazar software required. FPGA development tools are required as well for any FPGA changes or development, as described in section 1.7.2. However some example FPGA applications are available from AlazarTech and, if used as-is, do not require FPGA development.

2.3 Quick Start Guide

A Quick Start Guide is included in this document (Section 3). It is a step-by-step guide through the entire simulation, compilation and debug flow, to provide the user with a rapid startup and a hands-on overview of this Development Kit.

It is strongly recommended that the rest of this document also be read prior to initiating development, in order to understand the framework and start off in the right direction.

2.4 What is provided in the FPGA Development Kit

The FPGA Development Kit includes the following:

- AlazarTech JTAG Debug Board
- PCIe x8 1 meter extender cable
- Development Framework files, with sample design and documentation

The JTAG Debug Board provides the user with access to powerful debug tools, such as Altera SignalTap II®, In-System Memory Editor, the ability to quickly download new FPGA loads etc. It also provides LEDs dedicated to user debug functionality.

© 2003-2012 Alazar Technologies Inc.

The PCIe x8 extender cable allows the user to position the Waveform Digitizer board outside the host PC, thereby providing easier access to the JTAG Debug port, the signal ports, and visibility to the debug and onboard LEDs.

The Development Framework files include a design framework which takes care of the various chip interfaces and protocols so the user does not have to be concerned with these. It provides the user with a simple interface in which to insert custom functionality.

It also includes a simulation framework, which greatly reduces ramp-up time and allows the user to get to simulation results very quickly, and add the desired tests with minimal effort.

The FPGA compilation project and associated constraints files are also included, along with sample debug files for SignalTap® and the In-System Memory Editor.

Lastly, a sample design is provided to demonstrate where user functionality is inserted.

Note: An AlazarTech ATS9625 Waveform Digitizer board (sold separately) is also required to make use of the development kit.

2.5 Hardware Overview

The AlazarTech ATS9625 Waveform Digitizer board hosts a user-programmable FPGA, called the Coprocessor FPGA, which is connected directly to the Analog-to-Digital Converters and is entirely dedicated to the user's signal processing circuitry.

For more detail, please refer to section 4.1, as well as Figure 4-1 ATS9625 Board Block Diagram.

Two versions of the ATS9625 Waveform Digitizer board are offered for purchase (separately from the Development Kit). The difference is in the size of the FPGA device: the Standard version has a device large enough to suit most applications, while the High-Capacity version has a very large device suitable for very advanced and complex signal processing applications.

2.5.1 Coprocessor FPGA (CPF) Device Information

The Coprocessor FPGA located on the ATS9625 board is an Altera Stratix III device. These are high-speed devices providing a variety of resources, including logic, PLLs , RAM and specialized DSP/Multiplier blocks. The basic interface circuitry requires only a very small portion of the FPGA, the rest is completely available for the user's design.

There are currently 2 FPGA size options available when you order the ATS9625 Waveform Digitizer board:

1. Standard: EP3SL50F780C4 (~50k Logic Elements)
2. High-Capacity: EP3SE260H780C4 (~260k Logic Elements)

© 2003-2012 Alazar Technologies Inc.

Table 2-1 provides a brief summary of the FPGA's internal resources.

Table 2-1 Coprocessor FPGA Resources

Device	LEs	18x18-bit Multipliers (FIR Mode)	M144K RAM Blocks	M9K RAM Blocks	Total Embedded RAM Kbits	PLLs
EP3SL50	47.5k	216	6	108	1836	4
EP3SE260	255k	768	48	864	14688	12

Please refer to the Altera Stratix III Device Handbook (www.altera.com) for more detail on these parts.

2.6 Development Framework Overview

A complete development framework is provided by AlazarTech in order to facilitate the user's development effort. Out of the box, everything is provided and readily accessible to simulate the design, compile it, download it to the board and see it working. Extra content has also been provided to go through the design flow and demonstrate simulation and live debugging tools, in order to minimize ramp-up time and allow the user to get to results as quickly as possible.

The framework provided is described in the following sections.

2.6.1 Coprocessor FPGA Design Framework

The design framework takes care of the various chip interface ports, protocols specific to the board's chip-to-chip interfaces etc. It provides the user with a simplified, straightforward interface which allows the user to focus only on their custom design functions.

This significantly reduces ramp-up time and allows the user to get to results as quickly as possible.

2.6.2 Sample User Design

A sample design is provided to make the board fully functional right out of the box. It provides the user with a clear indication of where to insert their custom code, while also providing working code with typical expected functionality, for all functional blocks the user doesn't wish to modify.

The bare framework comes with a sample design that simply passes the data and trigger through the Coprocessor FPGA. Other, more complex sample designs may be available from Alazar, please contact us for more details.

© 2003-2012 Alazar Technologies Inc.

2.6.3 Quartus FPGA project

A fully working Quartus FPGA project is provided, including:

- I/O constraints
- Timing constraints (with Quartus' TimeQuest and .sdc files)
- SignalTap and In-System Memory Editor samples to show the user how these powerful debugging tools can be used.
- Customized Timing Report script which provides all relevant timing reports without having to start Quartus' TimeQuest.

2.6.4 Simulation Framework and Testbench

A complete simulation framework and testbench with extended data access, including:

- Arbitrary Waveform Data Generator: reads samples from user-defined input file. Allows the user to generate input signal using 3rd party tools, such as Matlab, live data capture, Excel, Text Editor, etc.
- Basic sine wave Data Generator
- Data Capture to file. Allows capturing FPGA simulation output to a file for further analysis.
- CPU read/write/verify access procedures: to access internal FPGA registers as the software does.
- Automatic verification capability: monitors testcase failures throughout the simulation and reports Pass/Fail at end of simulation.
- Stimulators for board-level devices (ADC's, clock oscillators etc.)
- Pre-configured wave file which displays all relevant signals in a logically organized fashion, including input and output data streams in digital and analog view.
- Script to compile the entire design and testbench, and configure Modelsim with a single command. This saves significant initial ramp-up effort.
- Script to start the complete simulation with a single command.

2.6.5 Development Environment Facilitators

These are Extras provided as is without support, but may be of benefit to the user:

- Quartus Incremental Compilation compatibility. (Saves recompile time for debugging, and/or for large projects)
- SlickEdit® Editor pre-configured project and workspace files. Useful for allowing single-key project compilation and simulator execution, project file navigation, keyword-based design navigation etc.

2.7 Pre-requisites

This section outlines the pre-requisites for developing using the FPGA Development Kit framework.

© 2003-2012 Alazar Technologies Inc.

2.7.1 FPGA Design Expertise

AlazarTech has put in significant effort to provide a framework that lets the user only focus on their actual data processing block without having to worry about the external FPGA interconnects, interface timings and other board-related intricacies.

Furthermore the entire development framework has already been created in order to make the learning curve as smooth as possible, to allow the user to get started and be productive as quickly as possible.

However it is necessary to recognize that the user will be developing FPGA circuitry. As such the user is required to have working knowledge and experience developing FPGA's, namely:

- Compiling FPGA's using vendor-provided tools, and understanding typical FPGA design constraints and warning messages from the tools.
- Designing and coding in VHDL.
- Debugging FPGA designs using industry-standard practices.
- Using Modelsim.
- Reaching Timing Closure on FPGA designs, and applying Timing Constraints using the .sdc (Synopsys Design Constraints) language.

The language used for the framework is VHDL, and as such the easiest user flow is to use VHDL for the user design blocks as well. Note however that it is possible for the user to develop using other languages (Verilog, schematics), but doing so will limit Alazar's support, may require more expensive tools (e.g. mixed-language simulation) and/or limit the framework's effectiveness. It is likewise for simulation.

2.7.2 Tools, Licenses and Equipment

The following tools, equipment and licenses are required to use the User Programmable Development Kit:

- Altera Quartus®: The supported version is 12.0, Service Pack 2. The user needs to purchase a license separately for this software.
- Mentor Graphics' Modelsim®: Any edition, including Altera Edition. The free Altera Starter Pack should be sufficient for small designs but is subject to limitations. The supported version is 10.0d.
- PC with spare PCIe x8 or x16 slot for the AlazarTech ATS9625 Waveform Digitizer board.

The following is not required but is strongly recommended, since it provides access to powerful debugging tools:

- Altera USB Blaster® (or equivalent)

The framework was developed on a Microsoft Windows® environment and the files provided are in the corresponding formats. The user should use that environment for the smoothest transfer and to have access to all the framework's features. Using another

© 2003-2012 Alazar Technologies Inc.

operating system should be possible but may not provide the full features of the framework, is not supported and has not been tested by Alazar.

2.7.3 Design Responsibility

The user is responsible for their own design, including the functionality, resource usage, timing constraints and timing closure.

As general FPGA design advice, it is worth noting that timing closure (meeting all timing requirements) typically gets more difficult as a device gets more utilized. As such the user should make the appropriate architecture and design decisions, and use appropriate design practices to ensure that their completed design will fit within the device while meeting timing requirements.

2.8 Contacting us

Please contact us if you have any questions or comments about this document, or the sample code.

Web	http://www.alazartech.com
Email	mailto:support@alazartech.com
Phone	+1-514-426-4899
Fax	+1-514-426-2723
Mail	Alazar Technologies Inc. 6600 Trans-Canada Highway, Suite 310 Pointe-Claire, QC Canada H9R 4S2

Note that you can download the latest drivers and documentation here:
http://www.alazartech.com/support/downloads.htm

© 2003-2012 Alazar Technologies Inc.

3 Quick Start Guide

The purpose of the Quick Start Guide is to show the user through the motions of the design flow. It is highly recommended that the Designer's Guide be read through before implementing a final design.

3.1 Installing the ATS962x Coprocessor FDK

Please run the setup.exe, which will extract all necessary files to the path you specify. It is recommended that you choose a location that will be suitable for your development.

Please note that you may copy the directory tree over to any other location for development if needed. It is highly recommended however that you copy the entire source code tree if you choose to relocate the source files.

3.2 Implementing your Design

For the sake of simplicity in this first walkthrough example, we will simply simulate and compile the existing design, unmodified.

3.2.1 Where to insert your design

The framework is built to accomodate your design as easily and seamlessly as possible.

The circuit design files should go in the `_source_files_user` folder. The user design's top-level hierarchical block is contained in the appropriately named "`user_top_level.vhd`". This file can be modified as desired, and design blocks and files can be added as needed.

On the simulation front, additional testbench source files should go in the `_source_files_user\sim` folder. The top-level file is `tb_coprocessor.vhd` and can be modified as desired, with functional blocks and files added by the user as needed.

Refer to the Design Description section of the Designer's Guide for complete details.

3.3 Simulating the Design

Simulation is an optional intermediate step which allows the user to debug design functionality much more quickly than compiling it and testing it in the lab.

Typically, setting up a simulation environment requires a significant amount of knowledge of the simulation tool to ensure libraries are created and mapped, the proper design files are compiled in the right order, the right simulation options are provided to the engine, an appropriate wave file is created, etc.

The FPGA Development Kit provides smart Facilitator scripts which allow the user to get going extremely quickly.

© 2003-2012 Alazar Technologies Inc.

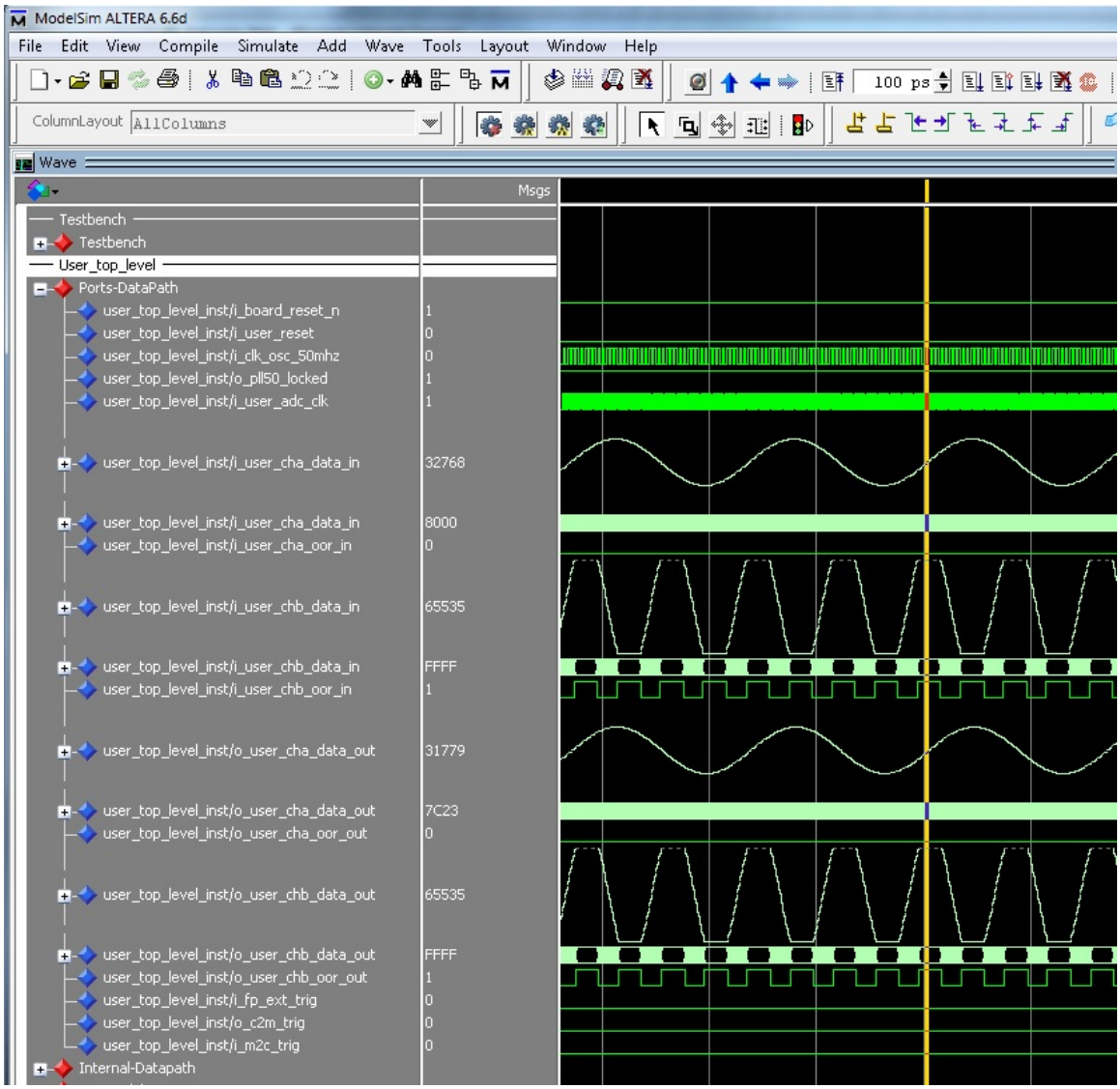
Here are the 3 simple steps to follow:

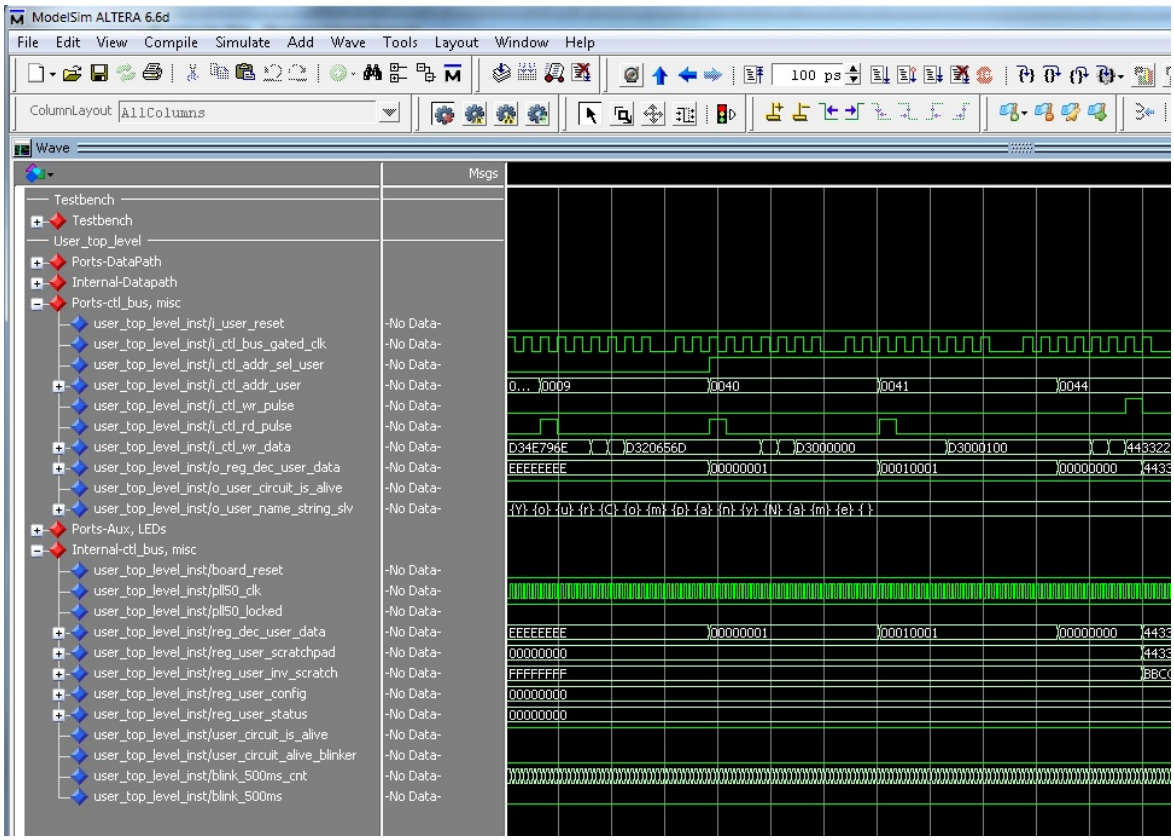
1) Start Modelsim and change the working directory to the `ats9625_cpf` directory, using ModelSim's File -> Change Directory... menu command.
2) In Modelsim's Transcript pane, type in "simbuild". This executes the simbuild.bat facilitator script which perform a number of tasks, including compiling the source files. There should be no error if the design was not modified.
3) type in "do simrun". This facilitator script performs a number of tasks, including starting up the simulation, calling up the supplied wave file, and running the simulation for the correct duration.

It's as simple as that! You can then observe the input and output signals on channels A and B of the sample circuit. You can also look at the various other signals, including triggers, auxiliary ports, control bus, etc.

The figures below demonstrate a typical sample output you can get from the simulation. Note: the actual waveforms shown below will differ from the simulation you will run; they are for illustrative purposes only.

© 2003-2012 Alazar Technologies Inc.


ModelSim ALTERA 6.6d
File Edit View Compile Simulate Add Wave Tools Layout Window Help
100 ps
ColumnLayout AllColumns
Wave
Msgs
Testbench
Testbench
User_top_level
Ports-DataPath
user_top_level_inst/i_board_reset_n 1
user_top_level_inst/i_user_reset 0
user_top_level_inst/i_clk_osc_50mhz 0
user_top_level_inst/o_pll50_locked 1
user_top_level_inst/i_user_adc_clk 1
user_top_level_inst/i_user_cha_data_in 32768
user_top_level_inst/i_user_cha_data_in 8000
user_top_level_inst/i_user_cha_oor_in 0
user_top_level_inst/i_user_chb_data_in 65535
user_top_level_inst/i_user_chb_data_in FFFF
user_top_level_inst/i_user_chb_oor_in 1
user_top_level_inst/o_user_cha_data_out 31779
user_top_level_inst/o_user_cha_data_out 7C23
user_top_level_inst/o_user_cha_oor_out 0
user_top_level_inst/o_user_chb_data_out 65535
user_top_level_inst/o_user_chb_data_out FFFF
user_top_level_inst/o_user_chb_oor_out 1
user_top_level_inst/i_fp_ext_trig 0
user_top_level_inst/o_c2m_trig 0
user_top_level_inst/i_m2c_trig 0
Internal-Datapath



Refer to Modelsim User's Guide and Reference Manuals for more details on using Modelsim.

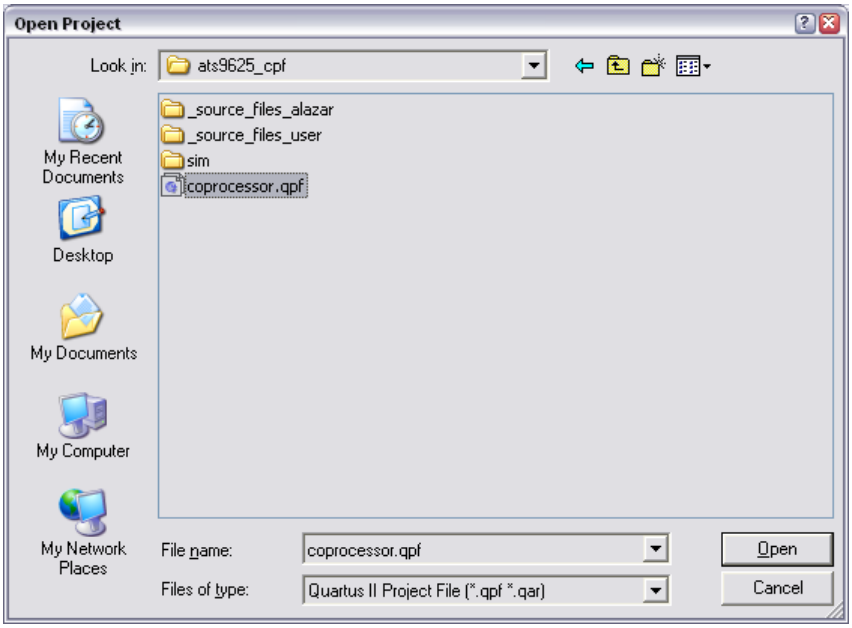
3.4 Compiling the Design and Producing a Downloadable File

To actually compile the design, one must install and start up Altera Quartus.

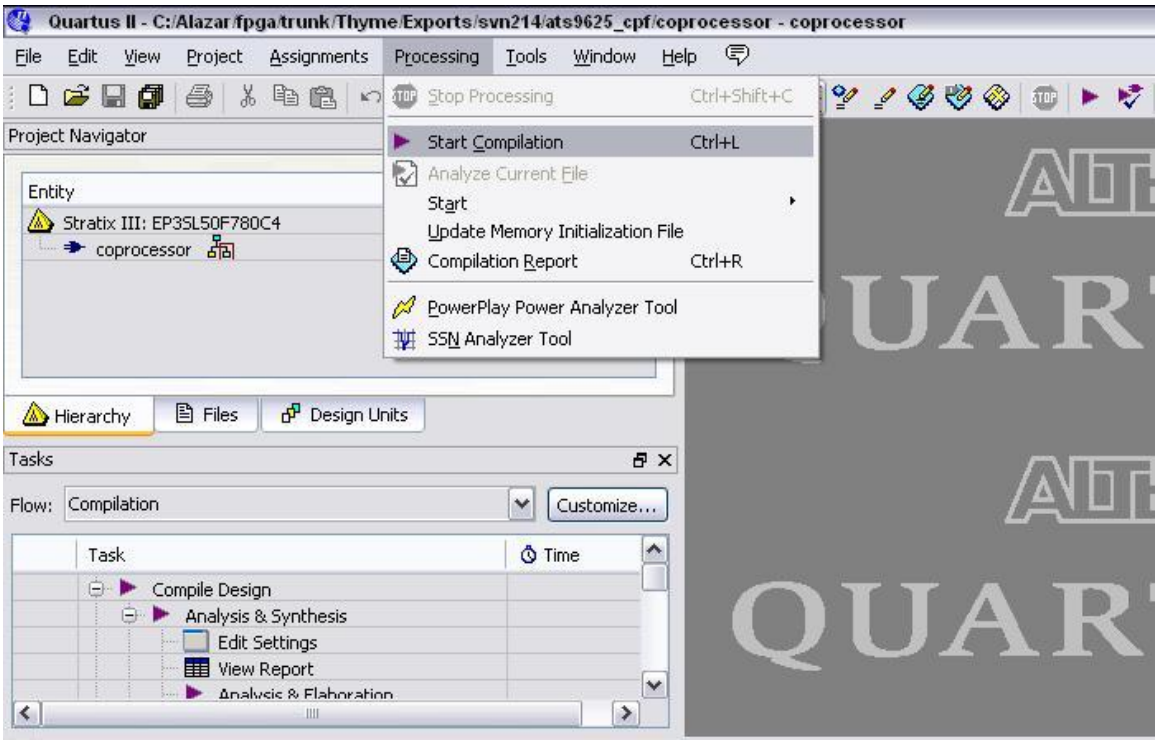
Once in Quartus, perform the following steps:

1) Open the Project: File -> Open Project.... The project is located in the `ats9625_cpf` directory, and is called coprocessor.qpf.

© 2003-2012 Alazar Technologies Inc.



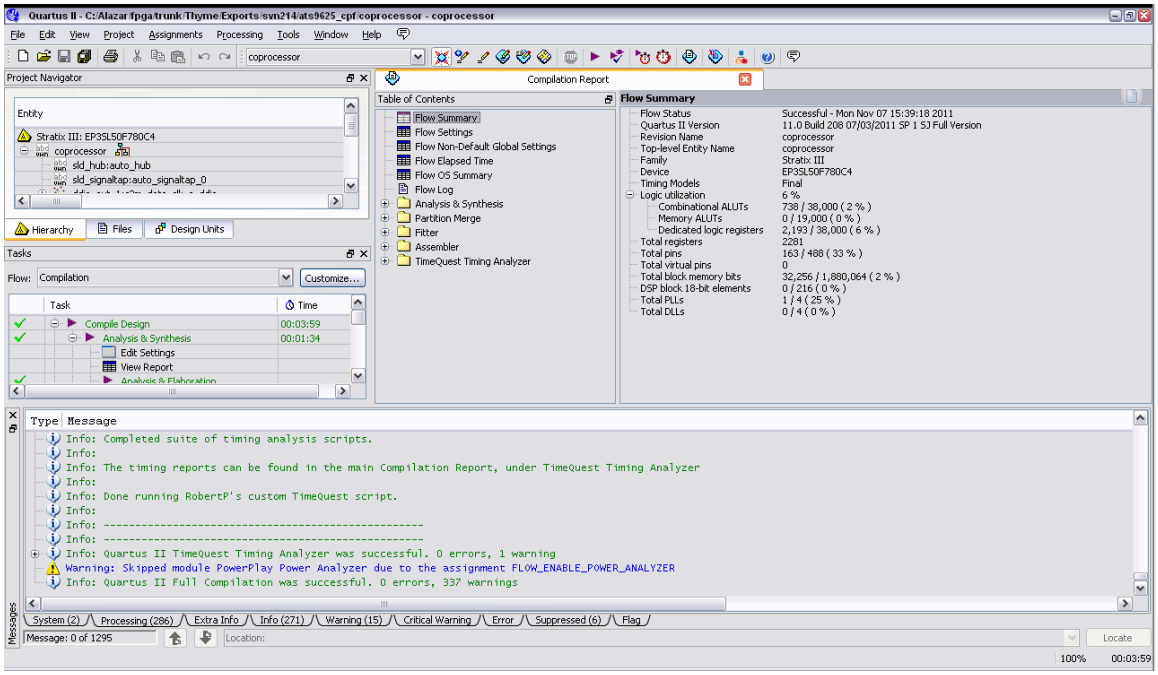
2) Compile the Design: Processing -> Start Compilation. Quartus will compile the entire design. This step may take a few minutes.



The FPGA is now compiled. This should be successful if the design supplied was not modified.

© 2003-2012 Alazar Technologies Inc.

You can go through the various Compilation Report sections to see the resuls of the compilation. Quartus' output files, including the FPGA programming files, are located in the same directory we opened the project in.


Quartus II - C:/Alazar fpga/trunk/Thyme/Exports/svn214/ats9625_cpf/coprocessor - coprocessor
File Edit View Project Assignments Processing Tools Window Help
Project Navigator
Entity
Stratix III: EP3SL50F780C4
coprocessor
sld_hub:auto_hub
sld_signaltap:auto_signaltap_0
Hierarchy Files Design Units
Tasks
Flow: Compilation Customize...
Task Time
Compile Design 00:03:59
Analysis & Synthesis 00:01:34
Edit Settings
View Report
Compilation Report
Table of Contents
Flow Summary
Flow Settings
Flow Non-Default Global Settings
Flow Elapsed Time
Flow OS Summary
Flow Log
Analysis & Synthesis
Partition Merge
Fitter
Assembler
TimeQuest Timing Analyzer
Flow Summary
Flow Status Successful - Mon Nov 07 15:39:18 2011
Quartus II Version 11.0 Build 208 07/03/2011 SP 1 SJ Full Version
Revision Name coprocessor
Top-level Entity Name coprocessor
Family Stratix III
Device EP3SL50F780C4
Timing Models Final
Logic utilization 6 %
Combinational ALUTs 738 / 38,000 (2 %)
Memory ALUTs 0 / 19,000 (0 %)
Dedicated logic registers 2,193 / 38,000 (6 %)
Total registers 2281
Total pins 163 / 488 (33 %)
Total virtual pins 0
Total block memory bits 32,256 / 1,880,064 (2 %)
DSP block 18-bit elements 0 / 216 (0 %)
Total PLLs 1 / 4 (25 %)
Total DLLs 0 / 4 (0 %)
Type Message
Info: Completed suite of timing analysis scripts.
Info:
Info: The timing reports can be found in the main Compilation Report, under TimeQuest Timing Analyzer
Info:
Info: Done running RobertP's custom TimeQuest script.
Info:
Info: ---
Info: ---
Info: Quartus II TimeQuest Timing Analyzer was successful. 0 errors, 1 warning
Warning: Skipped module PowerPlay Power Analyzer due to the assignment FLOW_ENABLE_POWER_ANALYZER
Info: Quartus II Full Compilation was successful. 0 errors, 337 warnings
System (2) Processing (286) Extra Info Info (271) Warning (15) Critical Warning Error Suppressed (6) Flag
Message: 0 of 1295
Location:
Locate
100%
00:03:59

Refer to the Altera website and manuals for more information on using Quartus.

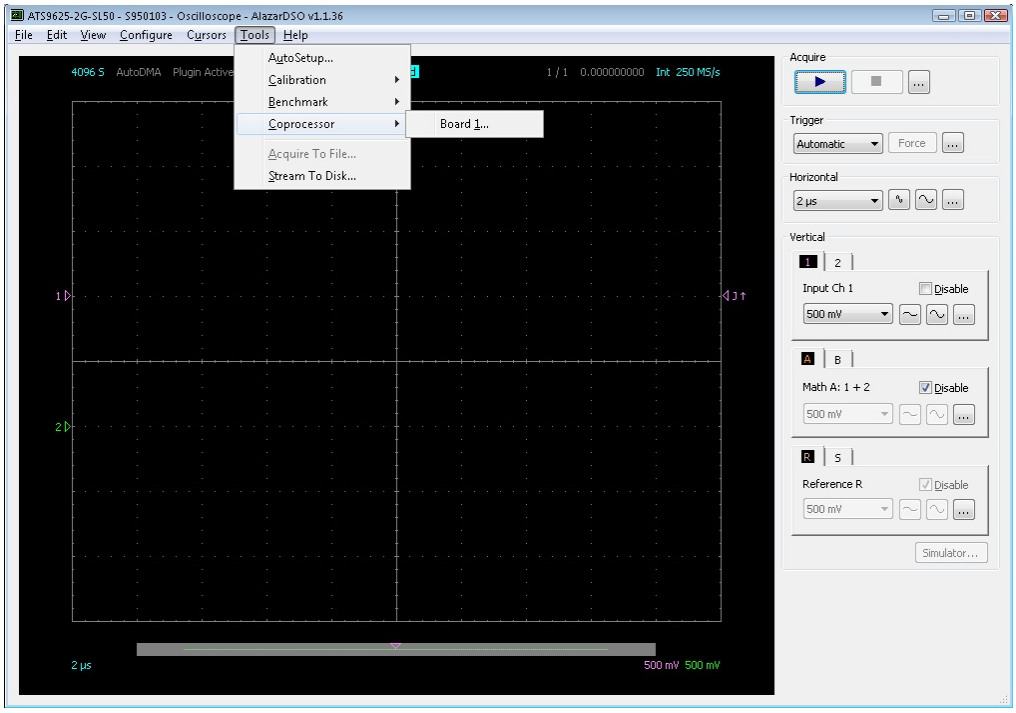
3.5 Downloading the FPGA to the Board

There are 3 options for downloading a new FPGA load to the board. The user can use whichever option is most convenient. They are described below.

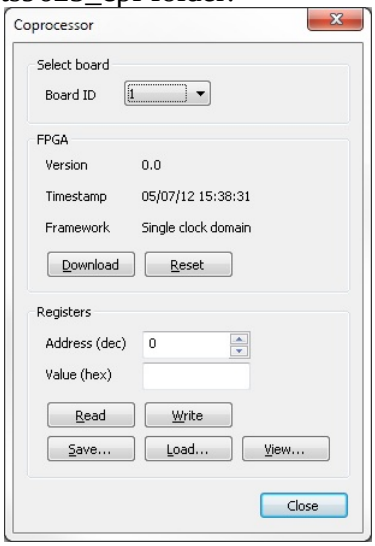
3.5.1 FPGA Download using AlazarTech DSO Software

This is the easiest method.

1) Once you have compiled your design in Quartus, simply start up AlazarDSO, and go in the Tools -> Coprocessor -> Board menu.

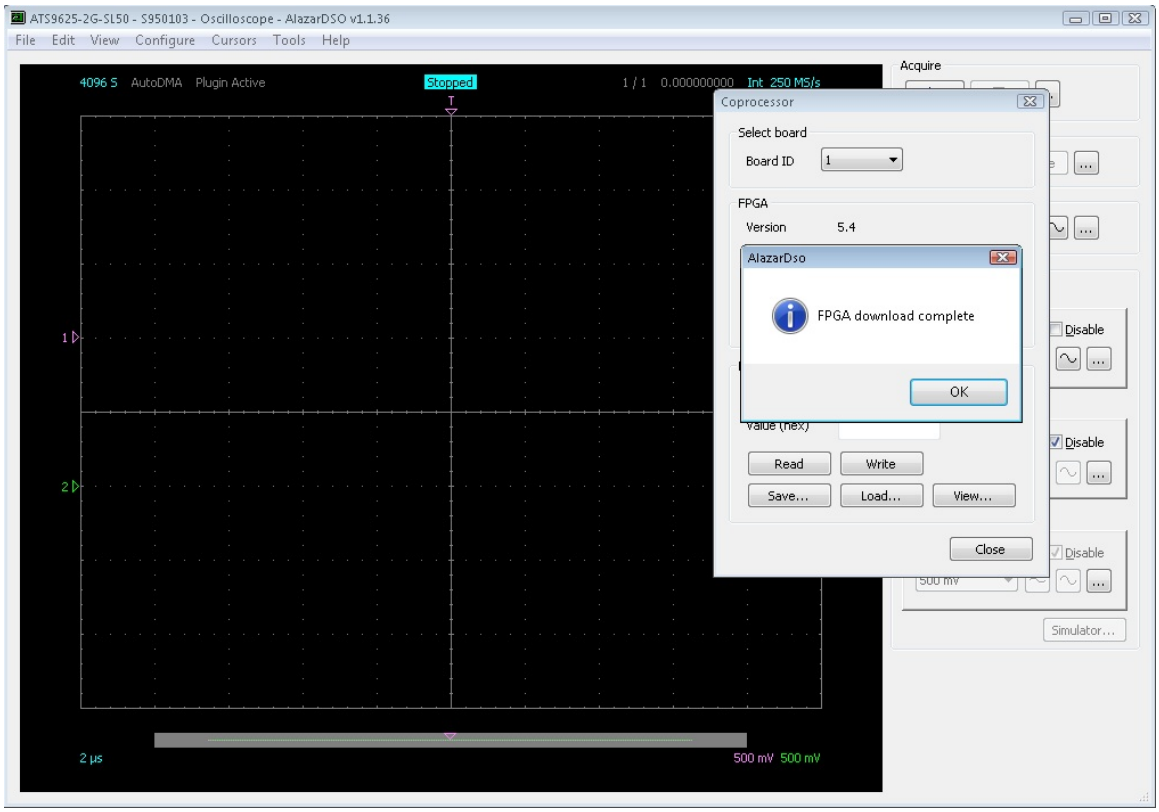


2) Once in the Coprocessor menu, click on the Download button, and select the coprocessor.rbf file from the ats9625_cpf folder.



Your FPGA will be downloaded immediately and a window will confirm the successfull operation, as shown below.

© 2003-2012 Alazar Technologies Inc.



3.5.2 FPGA Download using JTAG and Signal Tap II

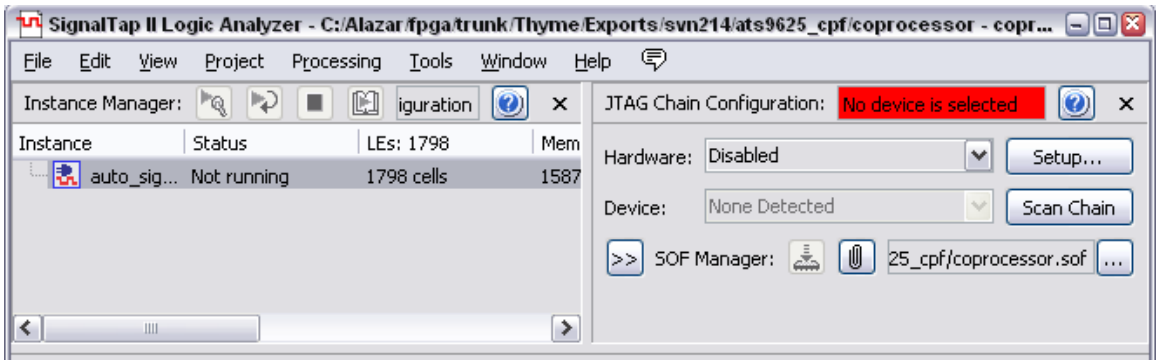
This method is typically used when using SignalTap to compile and debug the design. It requires Quartus to be installed on the PC, as well as an Altera USB Blaster connected to the ATS9625 board through the JTAG Debug board supplied in the FPGA Development Kit.

Note: In some versions of Quartus we have experienced inconsistent problems with the Quartus Programmer tool. However the SignalTap tool programs the FPGA successfully every time. We therefore recommend using the SignalTap tool to program the device via JTAG.

Open the Quartus project as described in section 3.4.

Then start the SignalTap II tool by selecting, from the Quartus menu, Tools -> SignalTap II Logic Analyzer. The SignalTap II window should open. Look for the JTAG Chain Configuration section at the top right of the window, as shown below.

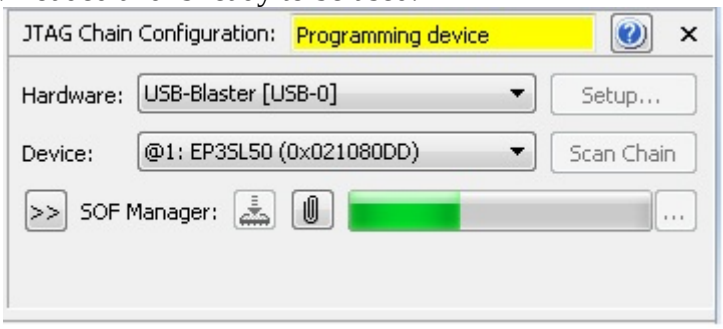
© 2003-2012 Alazar Technologies Inc.


SignalTap II Logic Analyzer - C:/Alazar/fpga/trunk/Thyme/Exports/svn214/ats9625_cpf/coprocessor - copr...
File Edit View Project Processing Tools Window Help
Instance Manager:
Instance Status LEs: 1798 Mem
auto_sig... Not running 1798 cells 1587
JTAG Chain Configuration: No device is selected
Hardware: Disabled Setup...
Device: None Detected Scan Chain
SOF Manager: 25_cpf/coprocessor.sof

1) Click the Setup... button, and select your USB Blaster. Note that the USB Blaster driver needs to be installed after installing Quartus. The driver can be found in the Quartus installation directory. Please refer to the Altera documentation and website for details.

JTAG Chain Configuration: JTAG ready
Hardware: USB-Blaster [USB-0] Setup...
Device: @1: EP3SL50 (0x021080DD) Scan Chain
SOF Manager: /ats9625_cpf/coprocessor.sof
Program Device

2) You may need to click the Scan Chain button, but usually th device gets found automatically.
3) Then click the "..." button to select the **coprocessor.sof** file in Quartus' working directory.
4) Lastly, click the icon with the blue arrow down into the chip to program the device. You should see a green progress bar, and the JTAG Chain Configuration status will show "Programming Device", as shown below. Once this is complete your now load has been downloaded and is ready to be used.


JTAG Chain Configuration: Programming device
Hardware: USB-Blaster [USB-0] Setup...
Device: @1: EP3SL50 (0x021080DD) Scan Chain
SOF Manager:

3.5.3 FPGA Download through software (ATS-SDK)

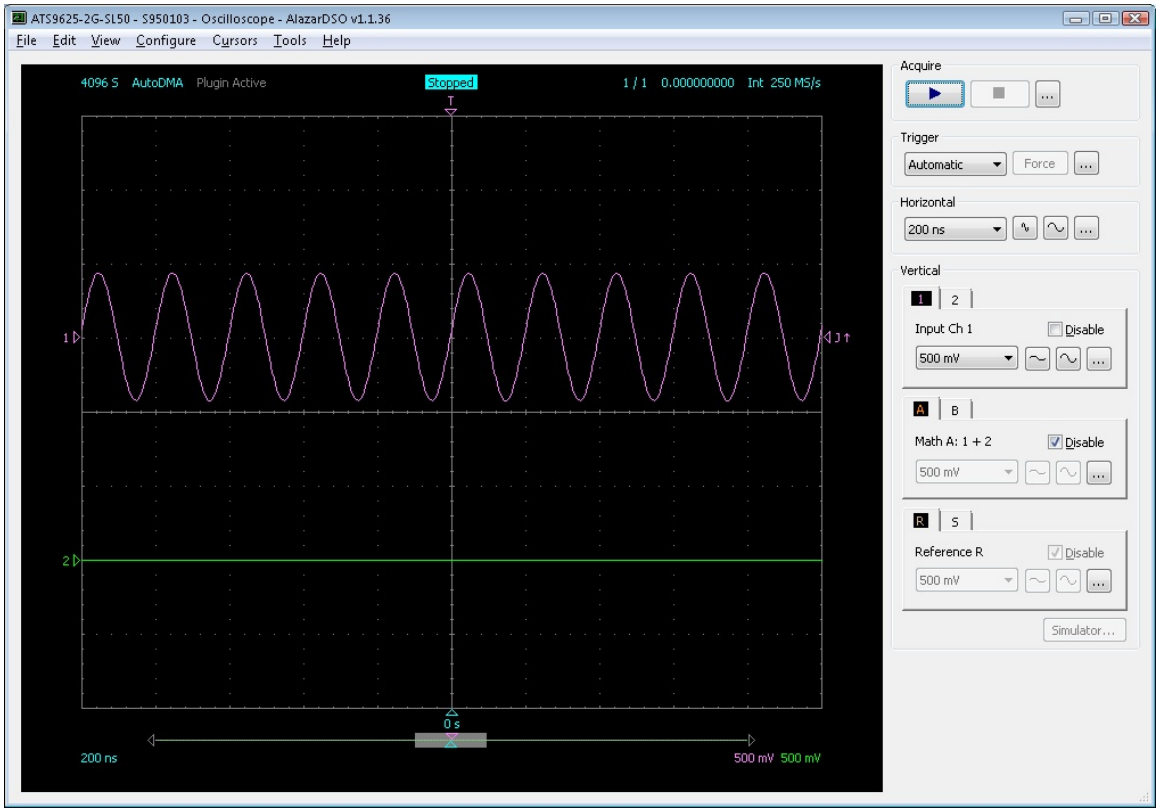
It is also possible for software to download the coprocessor FPGA. Please refer to the Software Development Kit document for more information about the method used to configure the coprocessor FPGA.

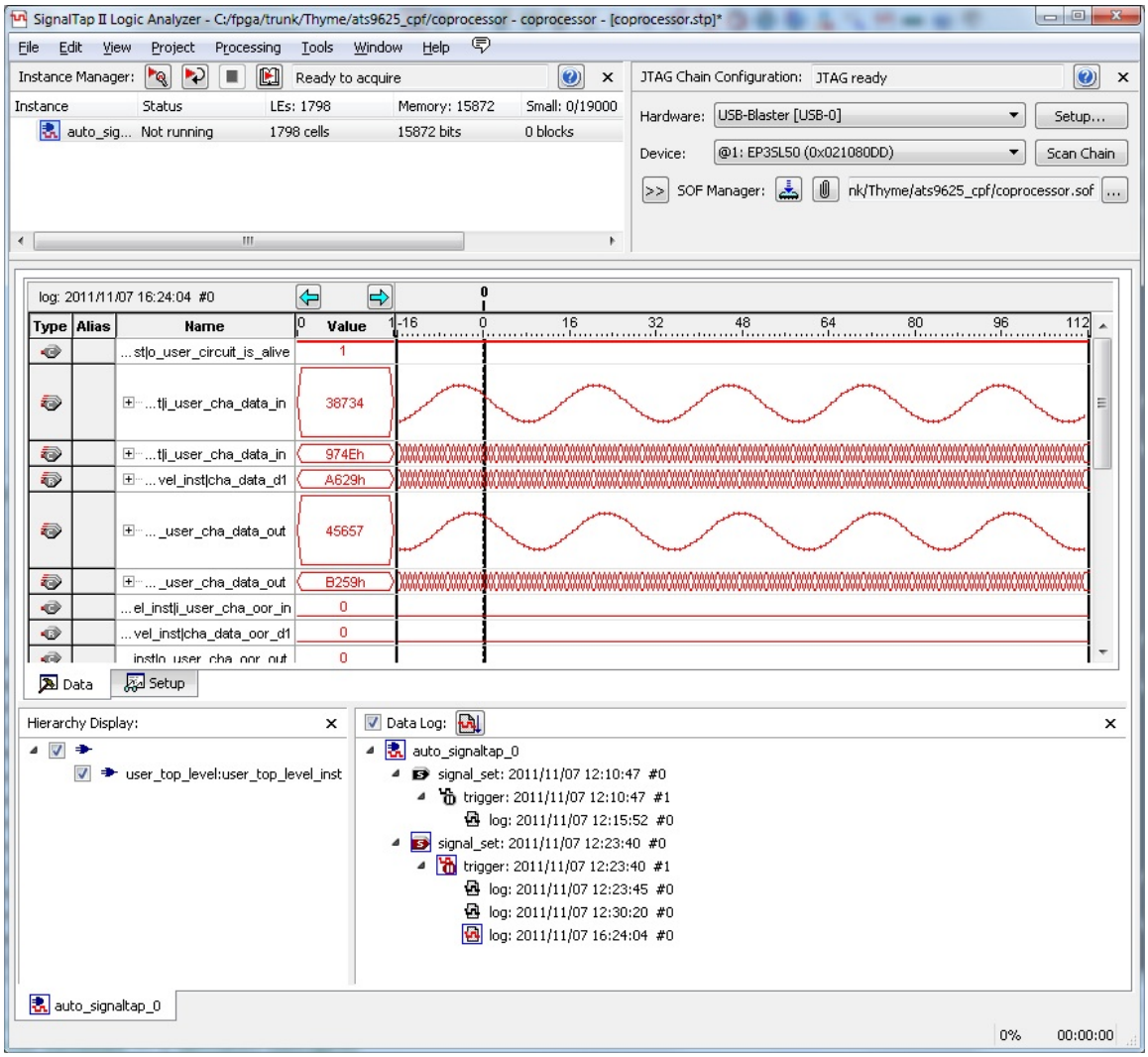
© 2003-2012 Alazar Technologies Inc.

3.6 Testing the User Design on a board

Once the new FPGA is downloaded, you are ready to test your new design. Make sure to supply a source waveform on the desired input channel(s), and verify acquisition through AlazarDSO and/or through SignalTap II, as desired.

Shown below are matching acquisitions using AlazarDSO and SignalTap II.





3.7 Design Modification Reminders

This serves as a reminder of files to add and modify when changing the design.

- Add new design files to simbuild.bat as well as to the Quartus design.
- Add the new signals to the waveform window in Modelsim, and save the new waveforms (typically as wave.do)
- Note on IP cores: IP blocks often have separate simulation models from their Quartus-compiled source files. Sometimes these are multi-file structures which need to be compiled separately from, or integrated into, simbuild.bat. Refer to the IP core's documentation and support on which files need to be compiled for simulation vs. compilation.

© 2003-2012 Alazar Technologies Inc.

4 Design Description

4.1 Architecture

4.1.1 ATS9625 Waveform Digitizer Board

The AlazarTech ATS9625 Waveform Digitizer board has 2 completely separate FPGA's for performing its functions. The user-programmable FPGA is the Coprocessor FPGA ("CPF") that is located between the Analog-to-Digital Converters (ADC's) and the main Alazar FPGA, as shown in Figure 4-1. The ADC input is AC-coupled on the ATS9625, and DC-coupled on the ATS9626.

The Coprocessor FPGA is entirely dedicated to the user's signal processing circuitry. Its only required function is to pass the data, triggers and auxiliary signals through to the main FPGA, as the user's purposes dictate.

The main Alazar FPGA is not configurable by the user. It takes care of all board functions, namely the PCIe interface, the data RAM buffering, interface to the Coprocessor FPGA, interface to software, housekeeping, etc.

This architecture allows the user to focus only on their desired custom functionality.

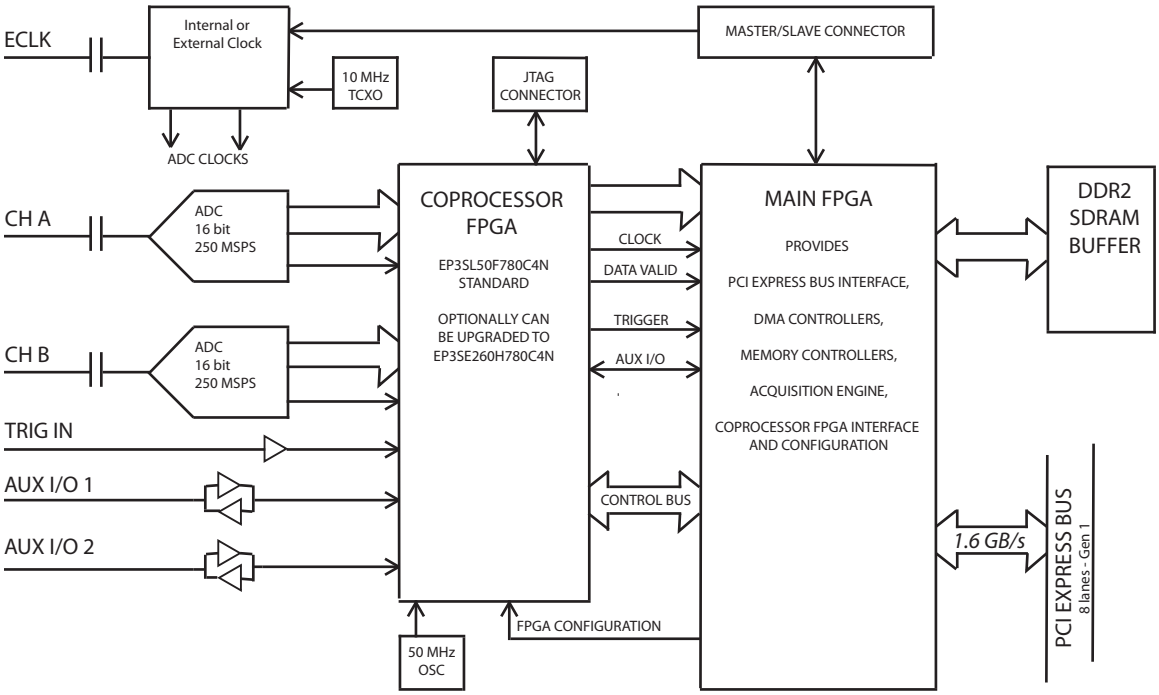


Figure 4-1 ATS9625 Board Block Diagram

4.1.2 Coprocessor FPGA

The Coprocessor FPGA is divided into 2 main sections: the Alazar section and the User section.

The Alazar section is a wrapper around the User section, and its function is to take care of the the various chip interfaces and protocols so the user does not have to be concerned with these. It provides the user with a simple interface in which to insert custom functionality.

The User only needs to be concerned starting at user_top_level interface and should not modify the Alazar files, including the top-level file.

4.1.3 File Folder Hierarchy

The FPGA Development Kit, once installed, will reveal the following directory structure:

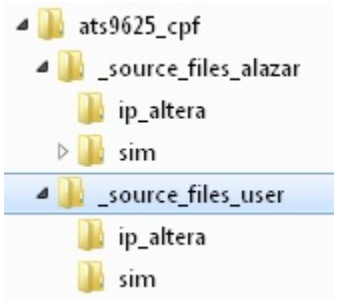


Figure 4-2 Package File Folder Hierarchy

Here is an overview of each folder in the hierarchy with its intended use.

`ats9625_cpf`: This is the top-level folder. It is the Quartus working directory, so all report files and output files will be located here, including the programming files. It is also the simulation working directory. Modelsim® is run from this directory as its working directory, and looks here to find some of its required files.

`ats9625_cpf\_source_files_alazar`: Alazar-controlled source files' location. The user should not modify the files contained here.

`ats9625_cpf\_source_files_alazar\ip_altera`: Alazar-specific Altera IP files. The user should not modify the files contained here.

`ats9625_cpf \_source_files_alazar\sim`: Alazar-specific simulation files. The user should not modify the files contained here.

`ats9625_cpf\_source_files_user`: user source files' location for the CPF design (as opposed to the simulation testbench files - see below). The user should add new files as required in this location.

© 2003-2012 Alazar Technologies Inc.

`ats9625_cpf\_source_files_user\ip_altera`: The user's Quartus Megawizard-generated files should be saved here. Since many files are auto-generated by the megawizard, it is cleaner to have them in their own directory.

`ats9625_cpf\_source_files_user\sim`: user simulation-related (testbench) source files' location. The user should add new files as required in this location.

4.2 Scope of Design Provided

The source files provided in the FPGA Development Kit include the entire Coprocessor FPGA design, including all physical I/O pins with their voltage levels and direction.

This is provided for simplicity in tools management, and to give maximum power to the customer to compile their own chip independently.

Care has been taken to separate the design into an Alazar section and a User section, such that the user does not need to be concerned about I/O's and specific protocols, and can focus on their design block.

4.2.1 User's Responsibility

Since the user has access to the complete FPGA design, it is the user's responsibility to exercise care not to break major functionality, or worse, cause physical damage to the ATS9625 board. The latter could happen if Input/Output directions were reversed for example, causing contention between output drivers on the board.

To prevent these problems, the user should not modify the alazar source files; they are in separate folder locations from the user source files.

It is also the user's responsibility to apply the proper timing constraints on their design. The existing Alazar circuitry has been constrained and the user need only change this according to their requirements.

4.3 Port Description - User Interface (user_top_level.vhd)

The following table describes the interface signals to the user_top_level.vhd module, which is the top-level file where the user places their custom function.

Table 4-1 User Port Interface Signals

Port	Direction	Description
i_cpf_reset_n	input	Asynchronous Coprocessor FPGA reset, active Low.
i_ctl_osc_50mhz	input	50MHz on-board clock oscillator, dedicated to CPF. Refer to section 4.9.2 for details.

© 2003-2012 Alazar Technologies Inc.

Port	Direction	Description
i_ctl_bus_gated_clk	input	Control Bus clock. Used only for software configuration and status registers. Refer to section 4.9.2 for details. NOTE: This clock is gated! It is only active during Control Bus R/W transactions.
i_user_adc_clk	input	ADC-supplied datapath clock, runs at sampling rate (nominal 250MHz). Refer to section 4.9.1 for details.
i_user_reset	input	Software-controlled reset signal for user circuit. Synchronous to i_user_adc_clk.
o_pll50_locked	output	pll50 locked status, directly from internal FPGA PLL. Mapped to status register.
o_user_circuit_is_alive	output	Hold High when user circuit has come out of reset and is functioning. Hold Low otherwise. Mapped to status register.
o_user_name_string_slv	output	Contains converted string of user name character ASCII values; Mapped to a register for reading via software.
Datapath-Input		
i_user_cha_data_in[15:0]	input	Channel A ADC Data, 16-bit, single data rate.
i_user_chb_data_in[15:0]	input	Channel B ADC Data, 16-bit, single data rate.
i_user_cha_oor_in	input	Channel A ADC Out-of-Range (OverRange) indicator (aligned with data)
i_user_chb_oor_in	input	Channel B ADC Out-of-Range (OverRange) indicator (aligned with data)
Datapath-Output		
o_user_cha_data_out[15:0]	output	Channel A Output Data, 16-bit, single data rate.
o_user_chb_data_out[15:0]	output	Channel B Output Data, 16-bit, single data rate.
o_user_cha_oor_out	output	Channel A Output Out-of-Range (OverRange) indicator (aligned with data)
o_user_chb_oor_out	output	Channel B Output Out-of-Range (OverRange) indicator (aligned with data)
o_user_data_valid	output	High to indicate that Data is valid; the same signal applies to both channels. Data is ignored by the Main FPGA when this signal is low.
Trigger Signals		
i_fp_ext_trig	input	External Trigger signal from FacePlate SMA connector
o_c2m_trig	output	External Trigger signal from Coprocessor FPGA to Main FPGA
i_m2c_trig	input	Trigger signal from Main FPGA to Coprocessor FPGA
i_m2c_armed	input	Indicates the Main FPGA is armed and ready to accept a trigger.
Auxiliary signals		
i_fp_aux_in_out[2:1]	input	Front Panel Auxiliary ports: Input value
o_fp_aux_in_out[2:1]	output	Front Panel Auxiliary ports: Output value (used if o_fp_aux_dir is set to drive output)
o_fp_aux_dir[2:1]	output	Front Panel Auxiliary ports: Direction (L = Front Panel to CPF, H = CPF to Front Panel)
i_mf_aux_in_out[2:1]	input	Main FPGA Auxiliary ports: Input value
o_mf_aux_in_out[2:1]	output	Main FPGA Auxiliary ports: Output value (used if o_fp_aux_dir is set to drive output)
i_mf_aux_dir[2:1]	input	Main FPGA Auxiliary ports: Direction (L = Main FPGA to CPF, H = CPF to Main)
LED signals		
i_m2c_led_on_n	input	Main FPGA LED output signal (propagates to Front panel through o_fp_led_on_n by default; can be overridden)

© 2003-2012 Alazar Technologies Inc.

Port	Direction	Description
o_fp_led_on_n	output	CPF to Front Panel LED Driver. (L= FP LED On; H= FP LED Off)
o_jtag_led_on[5:0]	output	6 LED/Testpoints located on the JTAG Debug board (when used), for the user's debugging purposes. (L= LED Off; H= LED On)
Control Bus		
i_ctl_addr_sel_user	input	Address Select for user memory section. (H = Address is within the User section; L = Address is within the Alazar section.) Note: keeps its last state until the next transaction.
i_ctl_addr_user	input	Address of current transaction on control bus.
i_ctl_wr_pulse	input	Write pulse; Indicates that write data is valid and should be captured on that cycle.
i_ctl_rd_pulse	input	Read pulse; for simulation clarity. Indicates when the read data is clocked in by output register.
i_ctl_wr_data[31:0]	input	Write data; validated by write pulse.
o_reg_dec_user_data[31:0]	output	Read data; provided by user circuit.

4.4 Datapath Functional Description

The datapath interface to the user_top_level module is a straightforward 16-bit data for each channel (Channel A, Channel B), one 16-bit sample per clock cycle. The same format is used on both the input and output side. The clock used is `i_user_adc_clk`.

Please refer to Figure 4-3 for a snapshot of the Datapath functional timing.

© 2003-2012 Alazar Technologies Inc.

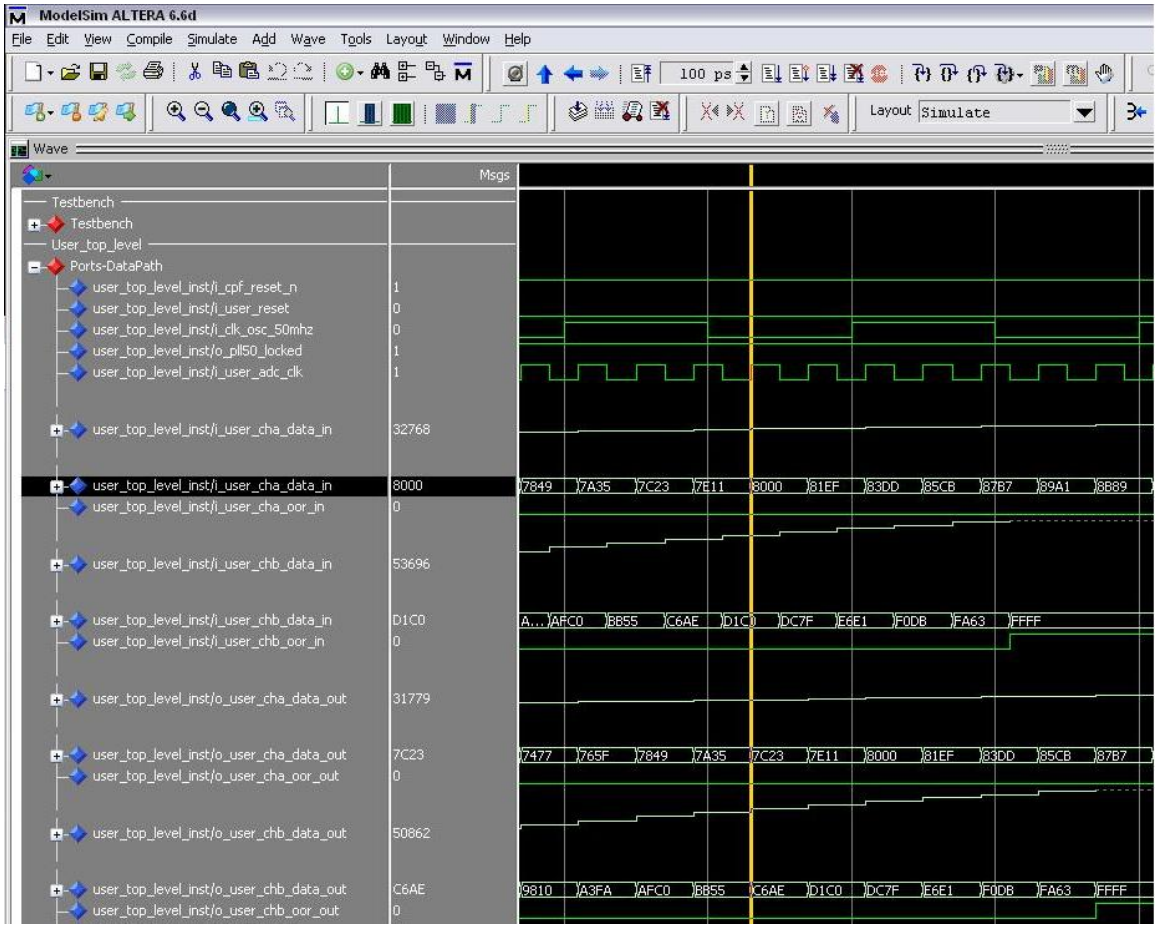


Figure 4-3 Functional Timing Diagram - Datapath

Coming into the user top level, we can clearly see from Figure 4-3 that there is one sample per clock cycle, coming in on the rising edge of the `i_user_adc_clk` clock, for both channels A and B.

Note that `i_clk_osc_50mhz` is not used in this design so is to be ignored for this example.

An Out-of-Range (OOR) situation is demonstrated on Channel B. This is relayed directly from the ADC's "OR" signal pin. Please refer to section 4.4.4 for more details on OOR.

4.4.1 Datapath/Trigger/Aux Latency Matching Considerations

In the provided example circuit, you may notice that there are only a few clock cycles of delay from data input to output in the example circuit, and the data is passed through unaltered. This will normally be significantly more for a user circuit with signal processing functionality. That is normal and expected.

However one consideration with respect to datapath latency is the use of Trigger and Auxiliary signals. A trigger signal that is passed through with a latency different than the data will result in acquiring a slightly misaligned window of data in a sample set.

Consequently, the user may need to match the trigger and auxiliary latency to the data latency, or regenerate them as necessary, within a certain tolerance. In some cases this tolerance is fairly large so is not a real concern. In other cases it is very important. That is for the user to determine based on their application, and is mentioned here as a reminder.

4.4.2 Trigger Pulse Requirements

The trigger signal going to the Main FPGA must meet the following requirements:
- `o_c2m_trig` must have a High and Low duration of at least 16 validated clock cycles. Clock cycles are validated by the `o_user_data_valid` signal. The 16 valid cycles do not need to be contiguous.
- Main FPGA data capture begins on the first validated clock cycle where `o_c2m_trig` is high

4.4.3 Data Format

The default data format coming from the ADC is 16-bit unsigned, binary offset at 8000h. The data format expected by the Main FPGA is this same format. Please note that the ADC can be reprogrammed to use other formats, but the main FPGA normally expects this format.

Other formats can be transferred but may need special management. Data format is indicated to the software through a data format register, found in the User section, Alazar-defined.

There is a Data Valid signal (`o_user_data_valid`) going to the Main FPGA, which indicates a sample as valid and allows it to be acquired. Samples with Data Valid low are not acquired. This is not utilized in the sample design; data is assumed valid on every clock cycle.

4.4.4 Out-Of-Range (OOR)

Additionally to the data, there are also out-of-range (OOR) indicators. On the input side, this originates from the ADC's to indicate that the input signal was of higher or lower voltage than its supported range. On the output side, the user is responsible for propagating this information, if desired.

Currently, the Main FPGA handles OOR in the following way: If there is at least one occurrence of OOR during an acquisition, this is latched and reported to software. The precise sample/time at which this occurred is not recorded, nor is the number of occurrences. Thus the user circuit does not have to align the output OOR perfectly with the output data, however if it is to be used, an ADC OOR occurrence must be propagated within the same acquisition window.

© 2003-2012 Alazar Technologies Inc.

4.5 Trigger Signals

The ATS9625 Waveform Digitizer board has a Trigger Input connector on its Front Panel. This signal is propagated on the board to the `i_fp_ext_trig` input port on the Coprocessor FPGA.

By default it is desirable to pass this signal through to the Main FPGA, via the `o_c2m_trig` output port. When the external trigger options are selected in software, the Main FPGA uses `o_c2m_trig` to start acquiring. The Main FPGA has no direct visibility of the `i_fp_ext_trig` input port; it relies solely on `o_c2m_trig`.

Propagation of the trigger signal as described above is already implemented in the sample design.

Please refer to section 4.4.1 for important latency considerations between the Datapath and the Trigger.

There is a 3rd port called `i_m2c_trig`. This is an input signal coming from the Main FPGA, and is reserved for future use.

4.6 Auxiliary Ports

The ATS9625 Waveform Digitizer board has 2 Auxiliary Input/Output connector ports on its faceplate. The user is provided full access and full control over these ports, and is provided with default software support.

4.6.1 Faceplate <-> Coprocessor FPGA

The Coprocessor FPGA (CPF) is the master driver and receiver for these ports (through on-board circuitry). The CPF has the following ports, reproduced here for convenience from Table 4-1:

Table 4-2 Faceplate to CPF Auxiliary Ports

Auxiliary signals		
i_fp_aux_in_out[2:1]	input	Front Panel Auxiliary ports: Input value
o_fp_aux_in_out[2:1]	output	Front Panel Auxiliary ports: Output value (used if o_fp_aux_dir is set to drive output)
o_fp_aux_dir[2:1]	output	Front Panel Auxiliary ports: Direction (L = Front Panel to CPF, H = CPF to Front Panel)

This gives the user full access to these ports for whatever custom use is required.

4.6.2 Coprocessor FPGA <-> Main FPGA

The Main FPGA has Auxiliary ports connected to the Coprocessor FPGA via a similar set of signals, reproduced for convenience here from Table 4-1:

Table 4-3 CPF to Main FPGA Auxiliary Ports

Auxiliary signals		
i_mf_aux_in_out[2:1]	input	Main FPGA Auxiliary ports: Input value

© 2003-2012 Alazar Technologies Inc.

Auxiliary signals		
o_mf_aux_in_out[2:1]	output	Main FPGA Auxiliary ports: Output value (used if o_fp_aux_dir is set to drive output)
i_mf_aux_dir[2:1]	input	Main FPGA Auxiliary ports: Direction (L = Main FPGA to CPF, H = CPF to Main)

The direction of the ports (`i_mf_aux_dir[2:1]`) is an input to the CPF, output by the Main FPGA, and so are the Auxiliary output values. They are user-controlled as well as monitored via software.

4.6.3 Typical Auxiliary Port Configuration

While the above provides maximum flexibility, the typical Auxiliary port behaviour is as follows.

In a typical configuration, the Main FPGA "thinks" it is connected to the faceplate ports directly. The CPF simply acts as a passthrough for the Auxiliary ports, and thus the Software behaves in a typical fashion by controlling and monitoring the Faceplate Auxiliary ports through the Main FPGA.

This functionality is already provided in the framework provided, so the user does not need to modify this, unless required by their custom functionality.

4.7 Control Bus

4.7.1 Control Bus Interface Functional Description

The Control Bus Interface is the hardware interface used for software to communicate control and status information to and from the Coprocessor FPGA, typically for register reads and writes. Note that the physical on-board inter-chip interface is all taken care of by Alazar-provided circuitry (i.e. included in _source_files_alazar) and translated to a simple protocol internal to the FPGA.

The Control Bus operates on a 32-bit non-maskable data width, which means that all 32-bits are always accessed on reads and writes. Each address location specifies one 32-bit location; it is not possible to do an 8-bit write, for example.

Please refer to Table 4-1, Control Bus section, for details on the control bus I/O ports to the user_top_level.
Read cycles currently have no wait states, thus read data is expected on the cycle following the address being valid, as indicated by the read pulse. (see below for a more detailed description.)

Please refer to Figure 4-4 below for 4 functional timing examples of the Control Bus. They are described in detail below.

© 2003-2012 Alazar Technologies Inc.

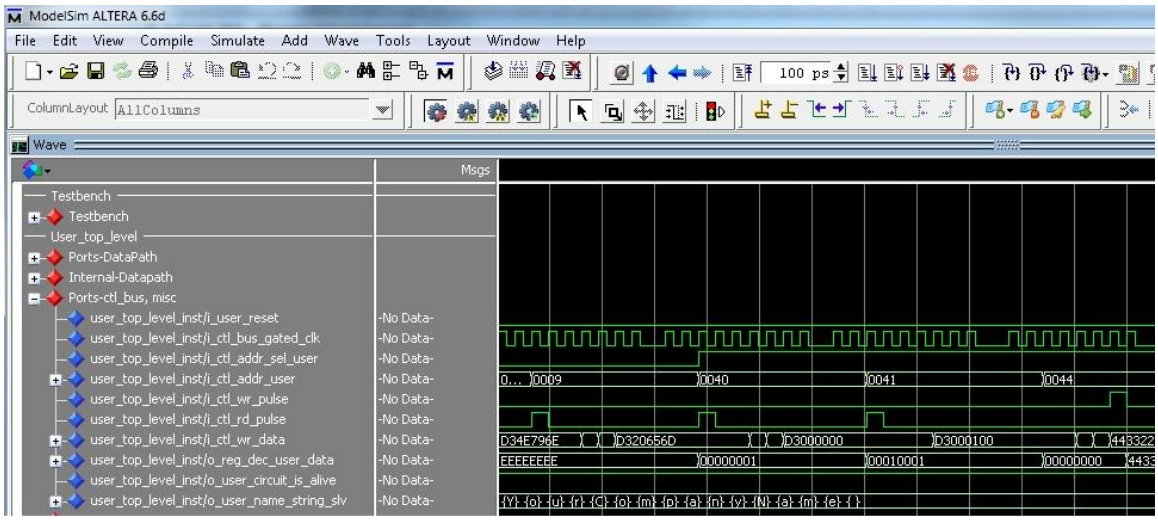


Figure 4-4 Functional Timing Diagram - Control Bus

The first transaction, at address 0009h, demonstrates a transaction which is NOT in the user section. Notice that `i_ctl_addr_sel_user` is Low, indicating to the user circuit that the transaction should be ignored by the user circuit.

The 2nd transaction, at address 0040h, demonstrates a transaction that is in the user section. Notice that `i_ctl_addr_sel_user` is High, indicating that the transaction is intended for the user circuit. The `i_ctl_rd_pulse` indicates that the read data (`o_reg_dec_user_data`) will be clocked in at the end of this cycle. Thus data is expected to be combinatorially decoded from by the address. Examples of how to do this are included in the `user_top_level.vhd` file, in the `reg_user_addr_decode_read` process. One can simply add registers to this process to expand the design.

The 3rd transaction is a user-section read at address 0041h, similar to above. It demonstrates that the `i_ctl_addr_user` does not necessarily transition between transactions; in fact once the transaction is finished, it keeps its last value until the next transaction.

The 4th transaction, is a Write to the user section. Again notice the `i_ctl_addr_user` does not transition. Notice the `i_ctl_wr_pulse` going high toward the end of the access. This pulse validates the write data and should be used to clock in the write data at the appropriate address. This is demonstrated in the VHDL code in the `user_top_level.vhd` file, in the `reg_user_wr_proc` process. One can simply add registers to this process to expand the design.

4.7.2 Memory Map Framework

A framework has been put in place to facilitate adding/removing registers and changing their addresses, with minimal work and minimal chance of errors.

`user_mem_map_pkg.vhd` contains a list of constants which define the address (or index: idx) location of each register, within the user section. Changing these constants to other values allows the user to relocate registers to different locations without having to update the individual read and write processes, and thus makes it a very quick change and reduces the risk of error (e.g. updating the read process but not the write, etc.) Furthermore it provides a clear and concise list of the register locations, all in one place.

Bit definitions and initialization (reset) values are also described under their register location.

Here is an example of how the memory map can look, with bit definitions and reset values for hypothetical registers:

```
constant c_idx_reg_user_signature          : natural                 := 32;
constant c_idx_reg_user_version            : natural                 := 33;
constant c_idx_reg_user_scratchpad         : natural                 := 34;
constant c_idx_reg_user_inv_scratch        : natural                 := 35;
constant c_idx_reg_data_format_cha         : natural                 := 36;
constant c_idx_reg_data_format_chb         : natural                 := 37;
constant c_idx_reg_samples_per_record_in   : natural                 := 38;
constant c_idx_reg_samples_per_record_out  : natural                 := 39;

constant c_idx_reg_user_cfg                : natural                 := 128;
  -- Init:  Select Triggered operation as startup default
  constant c_reg_user_cfg_init             : std_logic_vector(31 downto 0) := x"0000_0008";
  constant c_bit_cfg_wav_gen_sel           : natural         := 12;        -- x"0000_1000"
  constant c_bit_cfg_pre_trig_dsp_bypass   : natural         :=  8;        -- x"0000_0100"
  constant c_bit_cfg_out_u16_conv_bypass   : natural         :=  4;        -- x"0000_0010"
  subtype  t_bit_cfg_data_out_sel    is   natural range 3 downto  0;        -- x"0000_000x"
constant c_idx_reg_user_status             : natural                 := 129;
```

Figure 4-5 Memory Map Address Constants in user_mem_map_pkg.vhd

Going through an example for the hypothetical reg_user_cfg at address 128:

c_**idx**_reg_user_cfg:	The address, in decimal
c_reg_user_cfg_**init**:	The value this register should take on initialization (reset).
c_**bit**_cfg_wav_gen_sel:	A one bit location within this 32-bit register.
t_bit_cfg_data_out_sel:	A range of bits for a given n-bit field. Defined as a natural range type for easy access in the read/write process.

Please refer to the VHDL code for more detailed examples on the use of these constants.

4.7.3 How to Add New Registers

The read decoder and write decoder are already written in VHDL code, these processes:

- `reg_user_addr_decode_read`
- `reg_user_wr_proc`

New user registers can simply be added to it by first adding a c_idx_reg_user_xxx address per above, then adding them to the read and write processes in `user_top_level.vhd`, along with the registers functional processes.

The "scratchpad" and "inv_scratchpad" registers are coded as an example. Please refer to `user_top_level.vhd` for code details.

© 2003-2012 Alazar Technologies Inc.

4.8 Reset Signals, Startup Sequence, and FPGA Download

The main reset signal for the Coprocessor FPGA (CPF) is `i_cpf_reset_n`. This signal comes from the Main FPGA, is software-controlled and gets asserted after the CPF bitfile is downloaded, ensuring a safe and consistent startup sequence every time the CPF is downloaded, whether it be from a cold start or a subsequent update.

On the ATS9625 board, the Coprocessor FPGA (CPF) is the last device to be brought out of reset. This keeps things at their simplest for the user; no special startup sequence is required.

Internally to the FPGA, there is another reset signal (`i_user_reset`), which is intended for most customers' uses. Refer to the register's definition for details, section 4.11.1.4. This signal is directly controlled by a register internal to the CPF. It is asserted temporarily when i_cpf_reset_n is asserted (independently of the register's value), such that the user does not need to use the `i_cpf_reset_n` signal.

`i_user_reset` is also synchronized to i_user_adc_clk, which provides for a safe startup sequence coming out of reset, and aids the timing analyzer and timing-driven compilation.

Within the design, `i_user_reset` is split into 2 reset signals: `unconstrained_rst` and `fsm_rst`. This is done purely for timing closure reasons. At 250MHz, routing the reset on a global net fails Recovery timing. The solution is to split the reset into a timing-constrained net for the few recovery-timing-critical reset targets (counters, state machines), and all non-critical targets (datapath and pipeline registers etc) can use the unconstrained one.

Thus the `unconstrained_rst` should be used for the vast majority of registers, while the `fsm_rst` should only be used for specific registers that absolutely need to come out of reset all on the same cycle. `fsm_rst` is routed using local (non-global) routing resources, therefore the fanout must be kept small to remain routable. Additional distinct resets can be created if more fanout capacity is required.

4.9 Clocks

The Coprocessor FPGA receives multiple clocks, each with their own properties.

Please refer to section 5.3-"Clocking Scheme" for more information on which clocks to use under different circumstances.

4.9.1 user_adc_clk

This is the datapath's master clock. The user input data is in this clock domain, and the user output data is expected on this clock domain as well. This clock originates from the

© 2003-2012 Alazar Technologies Inc.

ADC, and thus runs at the ADC sampling rate, which is 250MHz by default (250MHz is also the maximum rate).

Note: By default, and unless the user chooses otherwise, this clock will be present and will run at 250MHz. However the AlazarTech ATS9625 board and Software provide many options to the user. In the various possible user configurations, this clock can be turned off, run at a lower frequency than 250MHz, or come from an external source. If you are expecting to make use of these advanced clocking options, please be sure to plan your use of clocks accordingly. Refer to section 5.3.

Refer to clk_osc_50mhz (section 4.9.2) for a clock that can be present and stable at all times.

4.9.2 clk_osc_50mhz

This comes from an on-board, independent clock oscillator that is dedicated to the CPF. This clock is always present when the CPF comes out of reset.

Should the user decide to not use it and wish to minimize noise, it is possible to turn it off via software configuration, in the Main FPGA. Upon restart, the oscillator's start-up time is approximately 10ms.

This 50MHz clock oscillator is *not* frequency-related to the ADC clock (250MHz nominal). One cannot assume, for example, that there are exactly 5 clock cycles of `user_adc_clk` in each `clk_osc_50mhz` clock cycle. FIFOs or equivalent structures must be used to pass data from one clock domain to another. Refer to section 5.3 for more information.

4.9.3 ctl_bus_gated_clk

This is the clock used by the Control Bus to transfer control information between Software and the CPF. The path for the transfers is from the Host CPU, through the PCIe interface, through the Main FPGA to the CPF. Similarly in the reverse direction.

WARNING: This is a gated clock. This means that the clock is not always running, and therefore any logic that is driven by it will freeze in time while the clock is inactive. It is not recommended to use this clock for anything else than software-accessible registers.

This clock is only active during read or write transactions, and is otherwise turned off. The reason it gets turned off is to minimize potential clock noise that could couple into the input signals.

The clock frequency is nominally 62.5MHz. It is guaranteed to be active throughout a ctl_bus access cycle, from the address valid to the read or write strobe. It can stop as quickly as 1 clock cycle after the write strobe, just enough time to register the data value.

© 2003-2012 Alazar Technologies Inc.

4.9.4 Other Clocks (Advanced)

4.9.4.1 enc_clk

This clock comes from the board's master clock synthesizer, and its frequency can be software-programmed. It has the same origin as the ADCs' clocks, so is frequency-related. It is not present on reset and requires software configuration.

4.9.4.2 c2m_data_clk_b

This is a Reserved-for-future-use clock on the CPF-to-Main FPGA datapath. Currently only one clock domain is supported for both Channels (`i_user_adc_clk / i_adc_cha_clk`).

4.10 LEDs and Debug I/O's

The single LED located on the Faceplate of the ATS9625 is driven by the Coprocessor FPGA through `o_fp_led_on_n`. In typical operation, it is recommended to simply pass through the Main FPGA's LED signal which is intended for the Faceplate, `i_m2c_led_on_n`. This functionality is already implemented in the provided CPF source code.

Additionally, 6 Debug LEDs are provided and dedicated to the user. These LEDs are present on the JTAG Debug board, and are directly driven by output pins from the FPGA. In other words the LEDs are present on the JTAG Debug board but are completely independent of the JTAG serial bus.

The CPF ports are `o_jtag_led_on[5:0]`, and are connected to the JTAG Debug board's LEDs labelled `LED105..100`, respectively. A High value turns the LED on, Low turns it off. The LEDs are all green in colour.

4.11 Memory Map

The Coprocessor FPGA is accessible by the host PC through the Main FPGA. The address space consists of 14 address bits, each addressing a 32-bit register. This provides a total of 16,384 individual 32-bit locations.

The memory map is divided into 3 global sections:

Table 4-4 Memory Map - Global

Addr (dec)	Section Name	Description
0 - 31	Alazar section (Reserved)	This is a fixed section which the user must not modify.
32-127	User section, Alazar-Defined	This section is driven by circuitry in the user section, but the definition and locations of the registers is defined by Alazar.

© 2003-2012 Alazar Technologies Inc.

Addr (dec)	Section Name	Description
128-16383	User section, Open	This is open for the user to modify freely.

4.11.1 Memory Map - Alazar Section (Reserved)

The Alazar section of the memory is a reserved section of 32 locations that the user must not change. Bit definitions or special functions are further detailed in subsections below, per register.

Table 4-5 Memory Map - Alazar Reserved Section

Addr (dec)	Register Name	RW	Description
0	cpf_signature	R	An Alazar-internal unique identifier for the Coprocessor FPGA. Do not change. (1234_5678h)
1	cpf_version	R	The major and minor version of the CPF Alazar framework code release. Do not change.
2	cpf_variant	R	Upper 2 bytes define the variant of the cpf framework used, namely Standard or Advanced. Year timestamp is also included in the lower 2 bytes.
3	compilation_timestamp	R	Date & Time timestamp which gets auto-updated everytime the design is compiled in Quartus. Provides a truly unique identifier (MM:DD:HH:mm)
4	scratchpad	RW	A 32-bit RW test register which has no connection to circuitry other than the scratchpad_inv register.
5	cpf_config	RW	The main CPF Configuration register.
6	cpf_status	R	The main CPF Status register.
7-11	-- Reserved for future use --		
12	scratchpad_inv	R	A read-only register which shows the inverted value of the contents of the scratchpad register.
13-31	-- Reserved for future use --	-	

4.11.1.1 cpf_version Register Description

Bit	Bit Name	RW	Rst	Description
31-16	<reserved>	R		Reserved.
15-8	cpf_ver_major	R		The 8-bit value of the major version.
7-0	cpf_ver_minor	R		The 8-bit value of the minor version.

The major and minor version of the CPF Alazar framework code release. Do not change. Please use the version register in the User section for your versioning purposes.

4.11.1.2 cpf_variant Register Description

Bit	Bit Name	RW	Rst	Description
31-16	cpf_variant	R		The 16-bit value of the cpf framework variant. 0000: Standard framework (cpf) 8000: Advanced framework (cpf_adv)
15-0	year	R		The 16-bit value of the build timestamp year version, in a hex-digit to dec-digit format.

For the "year" description and format, refer to the compilation_timestamp Register Description.

Example: 80002012h: Variant 8000 (cpf_adv), Build timestamp year 2012.

© 2003-2012 Alazar Technologies Inc.

4.11.1.3 compilation_timestamp Register Description

This register is automatically updated when compiling the design in Quartus, and provides a unique identifier for every build that is made. This is a convenient feature to differentiate between multiple debug compiles of the same version, moreover it provides the exact time at which the load was compiled so makes it easy to trace back to source code changes.

The format is as follows: it should be read out of the register and displayed in a 32-bit hex format. Then each byte holds a value that should be looked at as a decimal number. Thus by reading the register in its 32-bit hex value, a human reader can very quickly interpret the compile time. For clarification:

09151005h: September 15th, 10:05 (10:05am)
10281759h: October 28th, 17:59 (5:59pm)
01020001h: January 2nd, 0:01 (12:01am (1 minute past midnight))

Bit	Bit Name	RW	Rst	Description
31-24	month	R		Month (01-12).
23-16	day	R		Day of month (01-31)
15-8	hour	R		Hour (24-hour format) (00-23)
7-0	minute	R		Minute (00-59)

4.11.1.4 cpf_config Register Description

Bit	Bit Name	RW	Rst	Description
31-1	<reserved>			
0	user_reset	RW	0	Active High signal which gets synchronized to the i_user_adc_clk domain and goes to user_top_level. Should be used to reset user circuitry. Held active during board reset, auto-cleared a few cycles after.

4.11.1.5 cpf_status Register Description

Bit	Bit Name	RW	Rst	Description
31-2	<reserved>	R	0	
1	pll50_locked	R	0	Active High signal which indicates that the PLL fed by the external 50MHz clock oscillator is locked. Provided by user logic; disregard if you choose to not use this PLL.
0	user_circuit_is_alive	R	0	Active High signal asserted by the user logic to indicate that it is out of reset and running, PLL's locked, and any other prerequisites the user may deem important.

© 2003-2012 Alazar Technologies Inc.

4.11.2 Memory Map - User Section, Alazar-Defined

The user section, Alazar-Defined registers range from address 32 to 127 decimal. The circuitry for these registers is in the user section, but they are defined by Alazar so software can interpret the correct information depending on the nature of the cpf functionality. The address map is defined in VHDL in the file called "user_mem_map_pkg.vhd".

Table 4-6 Memory Map - User Customizable Section

Addr (dec)	Register Name	RW	Description
32	user_signature	R	A user-defined signature for the FPGA load, which can be used to identify different CPF designs and functionality. Update as needed. Default: 0000_0001h
33	user_version	R	The major and minor version of the User's design block. Update contents as needed.
34	scratchpad	RW	A 32-bit RW test register which has no connection to circuitry other than the scratchpad_inv register.
35	scratchpad_inv	R	A read-only register which shows the inverted value of the contents of the scratchpad register.
36	data_format_cha	R	A read-only register which shows Channel A's data format and width. This describes the data output from the cpf to the main FPGA.
37	data_format_chb	R	A read-only register which shows Channel B's data format and width. This describes the data output from the cpf to the main FPGA.
38	samples_per_record_in	RW	A software configurable register which instructs the CPF FPGA how many samples coming in from the ADCs are acquired per record. Can be useful for trigger-based record acquisitions.
39	samples_per_record_out	R	The CPF instructs the software as to how many samples per record will be output from the CPF to the Main FPGA. This can be the same as samples_per_record_in, but in some advanced applications it is not. It is the responsibility of the user's cpf FPGA circuitry to provide this information (usually based on samples_per_record_in).
40-127	-- Reserved for future definition by Alazar --	-	

4.11.2.1 user_version Register Description

The VHDL definition of this register is located in user_version_registers_pkg.vhd. Update as needed.

Bit	Bit Name	RW	Rst	Description
31-16	user_ver_major	R		The 16-bit value of the major version.
15-0	user_ver_minor	R		The 16-bit value of the minor version.

© 2003-2012 Alazar Technologies Inc.

4.11.2.2 data_format_cha/chb Register Description

This register is used by Alazar software to properly configure the Main FPGA's acquisition parameters. It is important that the values of these registers properly reflect the actual data formats used for the acquisition data to be properly transferred.

Bit	Bit Name	RW	Rst	Description
31	Endianness	R	0	0: Little Endian, 1: Big Endian Note: Currently only Little Endian is supported.
30-16	data_format_type	R	00	00h: Unsigned Integer 01h: Signed Integer 02h: Floating Point (per IEEE 754) 03h: Custom
15-0	data_format_width	R	16d	The number of bits on the c2m_data bus used up by each sample. 00h: Channel disabled 10h: 16 bits 20h: 32 bits

Currently Supported Data Formats:

0000_0010h 16-bit Unsigned Integer.
0000_0020h 32-bit Unsigned Integer.
0001_0010h 16-bit Signed Integer.
0001_0020h 32-bit Signed Integer
0002_0020h 32-bit Floating-Point (i.e. single-precision)

© 2003-2012 Alazar Technologies Inc.

4.11.3 Memory Map - User Section, Open (Customizable)

The user open section starts at address decimal 128 and goes up to the top of the available addresses (see table). This address map is defined in VHDL in the file called "user_mem_map_pkg.vhd" and can be customized by the user as desired.

Table 4-7 Memory Map - User Customizable Section

Addr (dec)	Register Name	RW	Description
128	user_cfg	RW	Suggested configuration register, which contains bits to configure the user circuit.
129	user_status	R	Suggested status register, which contains bits to monitor the status of the user circuit.
130-16383	-- For user's purposes --	-	For the user to define and use as needed.

© 2003-2012 Alazar Technologies Inc.

5 Design Recommendations

5.1 Where to insert your design

The framework is built to accomodate your design as easily and seamlessly as possible.

The circuit design files should go in the `_source_files_user` folder, as outlined in section 4.1.3 - "File Folder Hierarchy". The user design's top-level hierarchical block is contained in the appropriately named "`user_top_level.vhd`". This file can be modified as desired, and design blocks and files can be added as needed.

On the simulation front, testbench source files should similarly go in the `_source_files_user\sim` folder. `tb_coprocessor.vhd` is the testbench top-level file that can be modified as desired, with functional blocks and files added as needed.

5.2 Please do not modify Alazar source files!

The user and alazar source files were separated to give the user a user-friendly sandbox in which to create their custom design. The Alazar source files contain minimalistic functionality which basically connects the user_top_level.vhd ports to the FPGA's ports in a way that is as direct as possible, while removing the need for the user to deal with the intricate and sensitive details of the chip-to-chip interfaces.

As such, we strongly recommend that the user refrain from changing the Alazar source files, and reserve the right to refuse providing support when that is the case.

Please contact us if you believe there is missing functionality that would require you to edit the Alazar source files. We will consider integrating this additional functionality to the framework. Please refer to section 2.8 for our contact information.

5.3 Clocking Schemes

The ATS9625 Waveform Digitizer is highly flexible in its clocking options, and thus provides a number possibilities.

- The normal default operating mode is for the clock to be driven from an on-board oscillator which results in a very stable 250MHz sampling rate.
- The sampling rate can also be changed by the onboard clock generator to a lower frequency of the user's choice.
- An external clock can also be used to provide a custom reference, which is then used to generate a chosen sampling frequency.
- The acquisition mode can be continuous (streaming) or of fixed size.
- The 50MHz oscillator dedicated to the Coprocessor FPGA is on by default, but can be turned off via software, for example when the user wants the absolute best signal encoding quality.

© 2003-2012 Alazar Technologies Inc.

Due to all this level of flexibility, there is not a single unified clocking scheme that covers all the possibilities. The following subsections describe possible clocking schemes for various usage scenarios.

Please note that running a complete FPGA datapath at 250MHz is a high-performance application. It is quite feasible, as demonstrated by the example designs, but does require careful design practises. We have attempted to provide and demonstrate these as much as possible in the examples, but nothing replaces experience with FPGA design.

Alternatives to running at 250MHz include:

- Limiting the acquisition clock rate to a lower frequency (e.g. 225MHz, 200MHz), if it is compatible with the application.
- Running the datapath at half the frequency, but twice the data width.

5.3.1 Single-Clock Scheme (Recommended)

The recommended clocking scheme is the simplest one, which is to use `user_adc_clk` for everything except the control bus interface registers. The standard design framework uses this scheme and it is ready-to-go. This scheme has a few pre-requisites however:

- External Clock input, if used, provides a clean and stable, non-varying frequency throughout the acquisition.
- If the ADC encode frequency is changed by the user, the user also resets the Coprocessor FPGA after a frequency change.
- The user's circuit, once placed & routed in the FPGA, meets timing requirements at 250MHz (or the maximum encode frequency used).

The `user_adc_clk` is effectively the clock used by the ADCs, so it provides a fully synchronous single-clock system, operating at the sampling rate (nominally 250MHz). In this scheme, the 50MHz oscillator (`clk_osc_50mhz`)is not used, and could optionally be turned off via software.

The External Clock input can also be safely used with the single-clock scheme, as long as it is a clean signal, with a fixed frequency throughout acquisition. The frequency must not exceed 250MHz. Note that an externally-provided clock to the ATS962x ends up clocking the ADCs, which in turn output that clock to the Coprocessor FPGA, and gets to the user block as `user_adc_clk`. Thus when using the external clock in the single-clock scheme, this clock becomes the reference that drives the datapath from the ADCs through the Coprocessor FPGA to the Main FPGA.

Alazar was careful to implement an FPGA datapath that does not make use of any PLLs. This allows maximum flexibility to the user, in that no frequency-dependent design constraints or parameter entry is necessary. Please note that the user may need to reset the Coprocessor FPGA when connecting and disconnecting the external clock, in order to reset the user's Coprocessor FPGA logic (such as state machines etc) to a known state.

© 2003-2012 Alazar Technologies Inc.

As an example, if your application is Optical Coherence Tomography (OCT) using burst mode clocking, then you must use an advanced clocking scheme since the frequency varies throughout the acquisition.

5.3.2 Multiple Clock Schemes

If the pre-requisites in section 5.3.1 are not met, then a more complex clocking scheme must be used involving multiple clock domains. It is the user's responsibility to determine the correct approach required for their needs, and requires the corresponding level of expertise. The following gives general advice in this regard.

Externally-provided clocks have a wide variety of quality levels which are dependent on the external equipment used to provide it as well as the way it is used. Some factors to keep in mind:

- External clocks can typically be enabled and disabled, connected and disconnected at any moment, causing clock glitches during the transitions which can lock up or put parts of the circuitry in an unknown state.
- External clocks may have very high jitter, or changing frequencies. Internal FPGA timing requirements may not always be met, thus causing circuit misbehaviours or lock-ups.
- When using an external clock, it is usually best to limit the size and control the state of the circuitry which runs off the external clock. The majority of the circuit should run off of a reliable clock, such as the one provided by the 50MHz onboard oscillator.

A typical way to use an external clock would be the following:

- A small front-end circuit runs off the external clock (`user_adc_clk`), clocking the write-side of an acquisition FIFO.
- The majority of the user's circuit would run off `ctl_osc_50mhz`. To support streaming, the processing frequency needs to be equal or faster than the sampling frequency, so `ctl_osc_50mhz` would likely be fed to a PLL to multiply it up to a frequency higher than the maximum external clock frequency.
- The acquisition FIFO would be reset by the user's circuitry at the beginning of an acquisition, or at the very least once the external clock is guaranteed stable. The FIFO's level needs to be managed for overflow and underruns.

Alazar provides a separate example project based on a multiple-clock scheme. This project is documented separately.

5.3.2.1 50Mhz Oscillator vs. ADC Sampling Clock Domains

The user must keep in mind that the 50MHz clock oscillator is never frequency-related to the ADC sampling clock, in any mode of operation. In particular, even if the 50MHz is multiplied to 250MHz using a PLL, that 250MHz will be a slightly different frequency than the 250MHz sampling clock. Proper clock domain transfer circuitry must be implemented, typically through FIFOs.

© 2003-2012 Alazar Technologies Inc.

Note that it is NOT recommended to use a matched frequency as above (250Mhz & 250MHz), since the ADC's 250MHz sampling frequency could be slightly higher than the 250MHz derived from the 50MHz oscillator-based frequency. It is recommended to use a nominally faster frequency to be certain that data is read out of an acquisition FIFO at least as fast, or faster, than it is written in. Then use flow-control to manage its fill level. For example, 251MHz derived from the 50MHz clock oscillator and 250MHz as an ADC clock frequency.

5.3.3 PLLs

PLLs are available for use within the FPGA. None are currently used by the Alazar logic in the single-clock scheme, so they are all available to the user.

5.3.3.1 A Warning about using PLLs

A user may decide to use a PLL to multiply or divide a clock. If you choose to do so, please consider the following:

A PLL needs a very stable, low jitter clock source to lock and stay locked to. Also, PLLs are programmed with a defined input frequency. The Alazar-supplied datapath specifically does not use a PLL because the board allows the use of various sampling frequencies, and of external clocks. External clocks are not stable when connecting/disconnecting the connector, and depending on the source may have high jitter which would cause the PLLs to lose lock. This would cause highly unpredictable circuit behaviour and most likely crashes.

PLLs also take some time to lock. Please refer to Altera documentation.

If you always use an internal clock of a specific frequency, only then is it safe to use a PLL. We do not recommend using PLLs unless your design requires it and you have experience doing so.

5.4 Memory Initialization Files

5.4.1 RAMs and ROMs

When creating RAM and ROM memories in the Quartus Megawizard®, you have the option to prefill the contents with a memory initialization file, which can be in the .mif or .hex format.

If you choose to do so, please be aware that a relative path to the initialization file is written in the generated file. Since Quartus and Modelsim operate in the same root directories, this path effectively points to the same place, which is desired and simplifies things.

That is the reason why Modelsim and Quartus both have their working directory in the same place. If you operate Modelsim from a different working directory, you will get errors from it not finding files.

© 2003-2012 Alazar Technologies Inc.

File/folder locations are described in detail in section 4.1.3.

5.4.2 Complex IP Blocks

More complex IP blocks created by the Altera Megawizard, such as FFT, FIR filters, NCO, etc, typically generate .hex files which Modelsim needs to find in order to start the simulation.

This can often be problematic because the IP simulation model calls up memory files with no path, thus Modelsim needs to find them in its working directory, else it fails with an error similar to this:

```
** Error: (vsim-7) Failed to open VHDL file "***.hex" in rb mode.
```

This is further complicated by the fact that Quartus will prioritize the use of the .hex and .mif files that are located in the project base directory over the ones located in the actual IP directory. Therefore if old .hex files are present from an earlier simulation run, then one use Quartus' Megawizard to edit an IP file and then compiles in Quartus, the resulting FPGA load will have the old memory contents.

The solution to the combination of these problems is described below.

5.4.3 Scripts copying .hex and .mif files to Project directory

In order to circumvent this problem,the above-described problems, both the Quartus pre-compile script "quartus_pre_flow_script.tcl" and the Modelsim simrun.bat contains an embedded .tcl script which automatically finds all occurrences of *.hex and *.mif files and makes a local copy in the workingQuartus project folder. This ensures that all files will be found by Modelsim, and that they'll be up-to-date with new or updated IP files. It also ensures that these .hex and .mif files are up-to-date anytime a Quartus compile is run.
If your IP files are located outside the main folder's subdirectories (not recommended), then you must ensure the memory files are copied over and kept up to date, otherwise your simulation will not start, or your model parameters may not match your compiled FPGA parameters.

5.5 VHDL and Naming Conventions

The entire design and framework usesand example designs use VHDL 20022008 conventions, for maximum portability (vsto make use of the now-widely supported VHDL 2008) enhancements.

The editor spacing is set to 2 spaces per tab, tabs were converted to spaces throughout.

5.5.1 Prefixes

Entity ports have prefixes to discern them from local signals.

- i_ inputs
- o_ outputs

© 2003-2012 Alazar Technologies Inc.

- io_ inouts are only used for tri-state drivers

Port-naming abbreviations:

- c2m Coprocessor FPGA to Main FPGA
- m2c Main FPGA to Coprocessor FPGA
- fp Faceplate (of the ATS9625 board)
- mf Main FPGA

Other prefixes are used to differentiate signals from other types

- c_ constants
- v_ variables
- t_ type definitions

5.5.2 Suffixes

The following suffixes are used to provide more information on signals:

- _n: active low
- _p: indicates that the signal is typically a single-cycle pulse
- _m<n> m for metastability. Used for synchronizer registers in clock domain transfers. <n> is the stage number.
- _d<n> d for delayed, by <n> cycles.
- _e<n> e for early, by <n> cycles.

5.5.3 Block structures

Block structures are used throughout the design for concurrent statements. There is no functional reason for this. It is used simply to make concurrent assignments show up in structure-aware text editors' outline view. This is convenient as concurrent statements otherwise tend to get "lost" in the sea of processes and component instantiations. For example:

```
  ------------------------------------------------------------
  led_defaults : block
  ------------------------------------------------------------
  begin
    -- Propagate the main FPGA's LED output to the front panel.
    o_fp_led_on_n                    <= i_m2c_led_on_n;

  end block led_defaults;
```

© 2003-2012 Alazar Technologies Inc.

6 Software and System Considerations

From a control perspective, the CPF is accessible through both the AlazarDSO GUI and the ATS-SDK Alazar API's.

The following are some of the items that can be accessed through the software interface:
- Acquisition Configure / Start / Abort
- Coprocessor FPGA download
- CPF Control Bus read/write operations
- Auxiliary I/O control and monitoring
- 50MHz oscillator enable/disable

Please refer to AlazarDSO documentation, or as well as the appropriate AlazarTech Software Development Kit (SDK) Programmer's Guide for more details.

6.1 Acquisition Transfer Modes

Whether using the ATS-SDK or AlazarDSO, it is important to set the acquisition parameters appropriately for the user's application. Please refer to the software documentation for details on the settings.

One important parameter to consider when implementing a custom FPGA is the acquisition transfer mode. The user must decide which mode to use, and the decision will depend on how the Coprocessor FPGA operates during an acquisition.

There are 2 main categories, as described in the subsections below.

6.1.1 All Modes other than Coprocessor-Managed Mode

In these acquisition modes, the Main FPGA is responsible for detecting and managing the Trigger, selecting the correct sample data window with which to form records, counting records and determining the end of acquisition, etc. It is basically managing the entire acquisition.

These modes are the simplest modes to use, and should be used when possible. In these modes, the Coprocessor FPGA looks like an ADC front-end to the rest of the system.

For these modes to be used, the Coprocessor FPGA must fulfill the following conditions:
- It provides a continuous flow of data samples at all times (even when not acquiring, i.e. when i_m2c_armed is low).
  Note: This continuous flow of data is allowed to be rate-adapted, through the use of Data_Valid (i.e. Data_valid going low for one cycle every few cycles), however there is always data flowing and data is not halted for long periods through the use of Data_Valid.
- The data format is Unsigned 16-bit, zero biased at 0x8000 for each channel. One or both channels may be used.

© 2003-2012 Alazar Technologies Inc.

- There is no trigger or acquisition engine in the Coprocessor FPGA; the main FPGA is in charge selecting the window of samples, through the use of its trigger engine and configured record sizes and buffer sizes, etc.
- Example applications are:
  - FIR Filter
  - Decimating FIR filter
  - Digital Receiver
  - etc.

A counter-example is, if you are trying to select or generate and transfer specific windows of data from the Coprocessor FPGA, such as for an FFT, then these modes are not recommended and you must use Coprocessor-Managed mode. See below for more details.

6.1.2 Coprocessor-Managed Mode

In this mode of operation, the Coprocessor FPGA takes full responsibility for the acquisition, including Triggering and indicating what data to transfer to the software buffer. The Main FPGA's Trigger and Acquisition circuitry is bypassed, in a way that whatever data is provided by the Coprocessor FPGA gets passed through to the software buffer. Valid data from the Coprocessor FPGA is indicated to the Main FPGA through the use of the `o_user_data_valid` signal from the user_top_level.vhd module.

The data is transferred as a 32-bit format agnostic word, in other words the system does not interpret or care about the format of the data. Data concatenated from Channel A and Channel B is simply transferred to the buffer as a 32-bit word, in the same order as received from the Coprocessor FPGA.

In this mode, the Coprocessor FPGA:
- Must have its own trigger engine
- Manages the acquisition: it must delineate which samples to process, and count samples per record and records per buffers (if that is desired information).
- Provides only the selected data that should go into the software acquisition buffer.
- Example applications are:
  - Transforms such as an FFT
  - Special data formats are used, such as Floating-point.
  - Custom record format, such as a number of samples followed by a certain number of "information bytes", (e.g. average, timestamp, etc.)
  - Custom triggering engine

This is the only Alazar-approved method of transferring CPF-selected, fixed records of data through the system.

© 2003-2012 Alazar Technologies Inc.

6.1.2.1 Acquisition Control & Data Transfer Mechanism

In Coprocessor-Managed mode, the Coprocessor FPGA manages the triggering, sample selection, processing and flow. When using AlazarDSO, the acquisition is manually started and stopped by clicking the start and stop buttons respectively. When using an ATS-SDK, the user software is responsible for starting and stopping the acquisition. For the reader that is familiar with the other acquisition modes, the Coprocessor-Managed mode is in many ways similar to Continuous Streaming mode.

The coprocessor FPGA is provided with the "`i_m2c_armed`" signal, which goes high when software starts an acquisition, and goes low when software aborts the acquisition.

While i_m2c_armed is high, every clock cycle where the signal (`o_user_data_valid`) is high, the 32-bit word of data formed by (`o_user_ch`**`b`**`_data_out[15..0]`, `o_user_ch`**`a`**`_data_out[15..0]`) - little endian format - is transferred through the Main FPGA to the software buffer. The same signal is used for both rate-adaptation (for example in multiple-clock or decimating applications) and for data sample selection (to indicate valid data).

In Coprocessor-Managed Mode, the acquisition and data transfer mechanism is as follows:

A) User software configures acquisition parameters and Starts the acquisition.
B) i_m2c_armed goes High.
C) Data is received from the CPF, qualified by o_c2m_data_valid, by the Main FPGA and placed in a small hardware FIFO.
D) As new data comes in, the Main FPGA transfers data from this internal FIFO into the PCIe DMA Buffer.
E) *Once the PCIe DMA Buffer is full*, the Main FPGA launches a DMA transfer of the buffer into Host memory, in a Software Buffer.
F) Steps C to E are continuously and concurrently repeated in a streaming fashion.
G) Software decides to abort, i.e. stop, the acquisition (through the appropriate ATS-SDK function call, or with AlazarDSO's Stop button). An automated mechanism takes over in the Main FPGA which "pushes in" dummy data into the Main FPGA's FIFO in C, such that the remaining valid data get written into the PCIe Buffer (as well as subsequent dummy data if necessary). Once the PCIe buffer is full, the final DMA transfer into host memory is executed.
H) i_m2c_armed goes Low.
I) Software can now access the final buffer.

It is important to understand this mechanism, as it can have an impact in some acquisition scenarios, especially at the end of an acquisition. See below for more detail.

© 2003-2012 Alazar Technologies Inc.

6.1.2.2 End of Acquisition & Buffer Transfer Considerations

User software can decide upon the end of Acquisition mainly in multiple ways, for example:

1) Polling a custom-created user CPF register for a status or a counter value.
2) Counting units of data (e.g. samples, records, words, buffers) that have come through to the software buffers.
3) Any other method, such as timeout, human intervention etc.

Method 1 or 3 is recommended whenever possible, as it requires no special consideration for the final data buffer transfer. When Method 1 is used, software simply waits for the user's custom CPF register to be a certain value, the proceed to issue the Abort command which then pushes the remaining pipelined data through.

Method 2 requires special consideration in some very specific cases. For example, if software is expecting 10 buffers worth of data, and the CPF produces exactly 10 buffers worth of data and no more, i.e. data_valid is set low after the last data. In this case, software would have received 9 buffers of data and be waiting for the 10th, which would not come out since some of its data is waiting for new data to push it through the Main FIFO and into the PCIe DMA Buffer.

In such cases, there are multiple solutions, but the easiest one is to ensure the CPF pushes extra data through at the end to ensure the 10th buffer gets filled and transferred to Host memory. This extra data can be in the form of extra record(s), usually a simple solution, or a trailing number of data_valid's at the end.

The amount of extra data needed is as follows:

1) To clear the Main's FIFO, 2000 clock cycles of data_valid being High
2) For the PCIe DMA Buffer, that is a function of the user's chosen DMA Buffer size vs. the number of data words the CPF provided up to that point.

Another solution, for example, could involve setting an appropriate timeout for reading a buffer, such that if this length expires, one knows the acquisition must be at the end, and thus proceeds to do the Abort (which then pushes through the last remaining data into the buffer), and then fetch that last buffer of data.

6.1.2.3 Summary of Recommendations

For the simplest system-level operation, here are some recommendations for Coprocessor-Managed mode. Each option provides a valid way to terminate the acquisition and get all the data.

a) Provide a register in the CPF that determines the end of acquisition. User Software polls that register for an expected value, and upon detection initiates an acquisition abort.

- OR -

© 2003-2012 Alazar Technologies Inc.

b) Set the DMA Buffer size to an even multiple of data samples expected. At the end of acquisition, have the CPF provide enough extra data_valids to push the remaining data through the Main's FIFO.

- OR -

c) CPF provides a continuous stream of records during an acquisition, such that data always gets pushed through to Host memory.

© 2003-2012 Alazar Technologies Inc.

7 FPGA Compilation Environment

7.1 Overview

A complete FPGA Compilation environment is supplied to the user. All Quartus project-related files are provided such that the user can simply open the Quartus project and compile the design. All assignments and timing constraints have been taken care of, as well as additional tool files and scripts to make the user experience as easy as possible while making powerful debugging tools available at the click of a mouse.

This sections describes these files and debugging facilities.

7.2 FPGA type - Selecting the High-Capacity (HC) Revision

There are 2 versions of the board, one with the standard FPGA and one with a High-Capacity FPGA (refer to section 2.5.1 for details).

The Quartus project supports both versions by using what Quartus calls "Revisions". By default, the Quartus project compiles for the standard FPGA size. If you purchased the High-Capacity version of the board, then you will need to select the coprocessor_hc revision of the Quartus project.

Select the coprocessor_hc revision by choosing the Project -> Revisions... menu item, highlight coprocessor_hc, then Set Current.

When compiling for the coprocessor_hc revision, Quartus files uses and generates files with the "coprocessor_hc.*" names.

7.3 File Descriptions

This section describes the files found in the main directory (ats9625_cpf), which are used by Quartus for compilation. Substitute "coprocessor" for "coprocessor_hc" when using the coprocessor_hc revision, for the .qsf, .rbf, .sof, and most output files. The.qpf, .sdc, .stp, and .tcl files are common to both revisions.

7.3.1 Quartus Project Files (.qpf, .qsf)

The Quartus master project file is "coprocessor.qpf". You can open the project by executing the Quartus command Open Project... under the File menu.

Quartus also makes extensive use of the "coprocessor.qsf" file, which specifies all project, flow and I/O constraints.

7.3.2 FPGA Programming Files (.rbf, .sof)

There are two FPGA programming files that result from a successful FPGA compilation in Quartus.

© 2003-2012 Alazar Technologies Inc.

7.3.2.1 coprocessor.rbf

The .rbf file is an Altera format; it is the format required for downloading the Coprocessor FPGA using the AlazarDSO.

7.3.2.2 coprocessor.sof

The .sof is also an Altera format; it is the format required for downloading the Coprocessor FPGA using the JTAG Debug board and the Altera USB Blaster®.

7.3.3 Timing Constraints file (.sdc)

This file is in the .sdc format (Synopsys Design Constraints) as required by Quartus' TimeQuest module. It contains all required timing constraints to fully constrain the design as released to the user.

The user must update this file as required, for example if PLLs are used etc.

7.3.4 Quartus Custom Script files

Some script files were created to improve functionality and user-friendliness.

7.3.4.1 custom_tq_script.tcl

This tcl script is automatically run at the Timing Analysis phase of the compilation. The default Timing Analysis that is performed by Quartus is of limited use as it does not provide enough detail when timing is not met.
This custom script performs a detailed Core Timing and I/O Timing analysis, as well the the default Summary analysis. This provides very useful information for the user to understand and isolate timing failures, without needing to open up TimeQuest and figure out how to analyze with more detail.

7.3.4.2 quartus_pre_flow_script.tcl

This script is run everytime the user compiles the design. It currently auto-generates the compilation_timestamp_pkg.vhd file which contains the date and time of compilation. These get mapped into a user-readable register (compilation_timestamp).

Thus, when using the ATS9625 board, the user can always read that register to know exactly when the CPF design was compiled, which in turn can help identify which version is being used, additionally to the manually updated version registers.

© 2003-2012 Alazar Technologies Inc.

8 Board-Level Debugging Facilities

8.1 AlazarDSO User Interface

The AlazarDSO software provides simple debug access to the Coprocessor FPGA. Through the menu shown in Figure 8-1, one can access the window shown in Figure 8-2. This menu provides access to the following:

- Shows the Coprocessor FPGA Alazar signature and version
- Shows the Coprocessor FPGA User signature, version and Timestamp.
- Allows the user to download a new coprocessor.rbf file
- Allows the user to read and write directly to the Coprocessor FPGA.

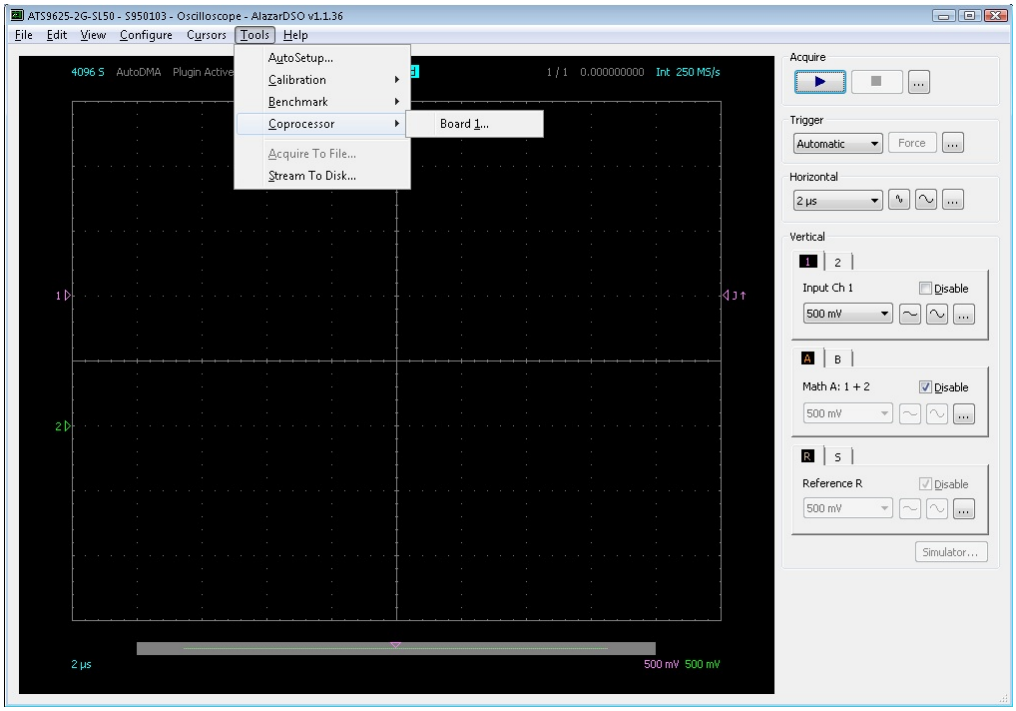


Figure 8-1 AlazarDSO Coprocessor FPGA Menu

Figure 8-2 AlazarDSO Coprocessor Window

8.2 AlazarTech JTAG Debug Board

This small board has 2 connectors and is included with the FPGA Development Kit. This board has 2 independent functions:

1. It has 6 LEDs which are directly connected to Coprocessor FPGA pins (independently and not related to the JTAG port itself). The pins are `o_jtag_led_on[5:0]`. The user's circuit can drive these LEDs as desired for debugging purposes.
2. It gives access to the Coprocessor FPGA's built-in JTAG port, and has a compatible connector for an Altera USB Blaster® or equivalent. This opens up the multiple debug facilities supported by Altera, such as SignalTap II® , In-System Memory Content Editor®, etc.

The JTAG Debug Board is connected at the top-left of the ATS9625 Board, when looking at the back side of the board. This is shown in Figure 8-3 below.

NOTES:
1) Ensure the USB Blaster connector is in the correct orientation (shown below).
2) Ensure no mechanical strain is applied to the board connectors or they could disconnect over time.

© 2003-2012 Alazar Technologies Inc.

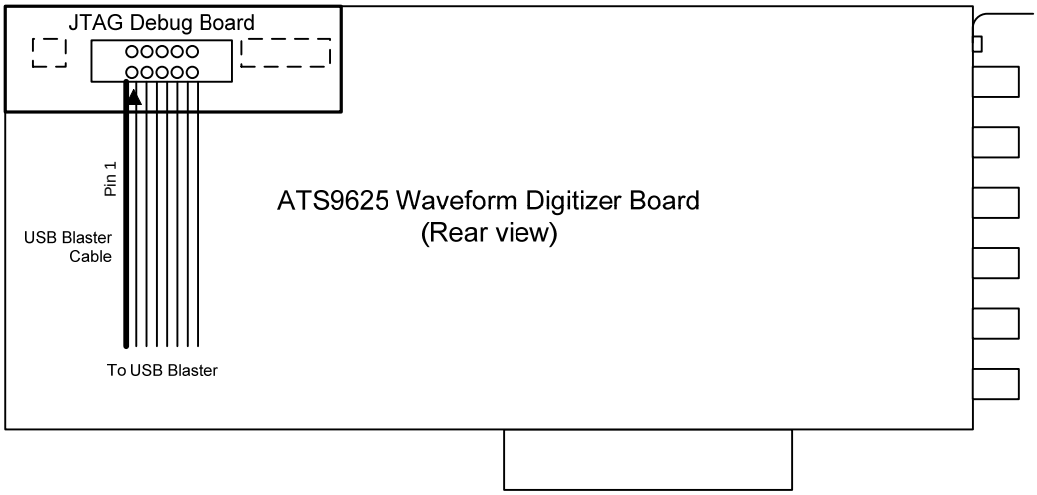


Figure 8-3 Connecting the JTAG Debug Board

8.3 Altera SignalTap II®

Altera's Signal Tap II® is a powerful live debugging feature of Quartus which allows the user to instantiate what is effectively a logic analyzer within the FPGA itself. Its interface is through a tool window in Quartus, via the USB Blaster and JTAG Debug Board, to the FPGA being debugged.

A pre-configured file (coprocessor.stp) is included to allow the user to immediately have access to the top-level datapath, trigger, auxiliary and LED signals without having to spend any time. This also serves as a good framework for the user to add further signals to it and customize it further.

A sample live-captured output from this preconfigured coprocessor.stp file is shown below.

© 2003-2012 Alazar Technologies Inc.

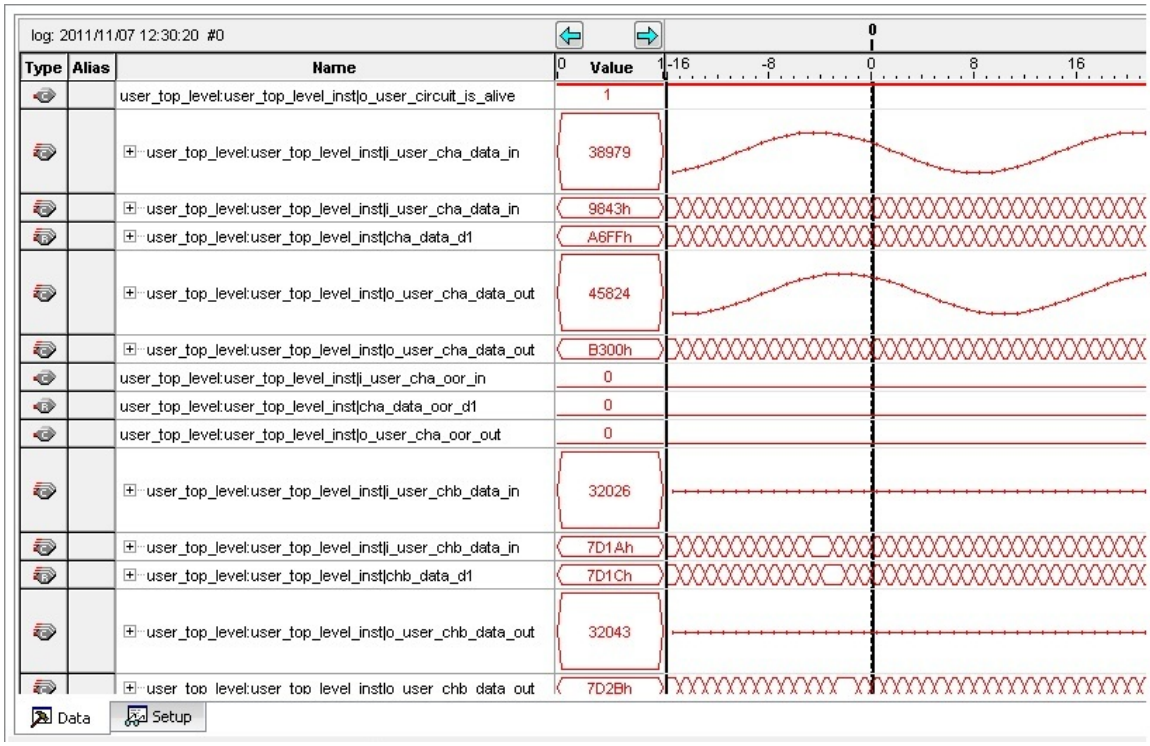


Figure 8-4 Signal Tap II® Sample Display

As you can see it is capable of displaying the signal graphically as well as in various radix formats, which are useful. Please refer to the Quartus manual for further information on Signal Tap II®.

8.4 Altera In-System Memory Content Editor®

The Altera In-System Memory Content Editor® is another very useful debugging tool. It allows read and write access to internal FPGA memories (RAMs and ROMs), such that the user can view and edit internal memory contents at will. Its interface is through a tool window in Quartus, via the USB Blaster and JTAG Debug Board, to the FPGA being debugged.

Note that for this feature to be accessible, the feature must be enabled when creating the RAM block from the Quartus Megawizard. The feature requires an extra read/write port so this can only be added to a ROM or single-port RAM. The sample design contains a Test RAM present for the sole purpose of demonstrating this feature. A sample live-captured output is shown below.

© 2003-2012 Alazar Technologies Inc.

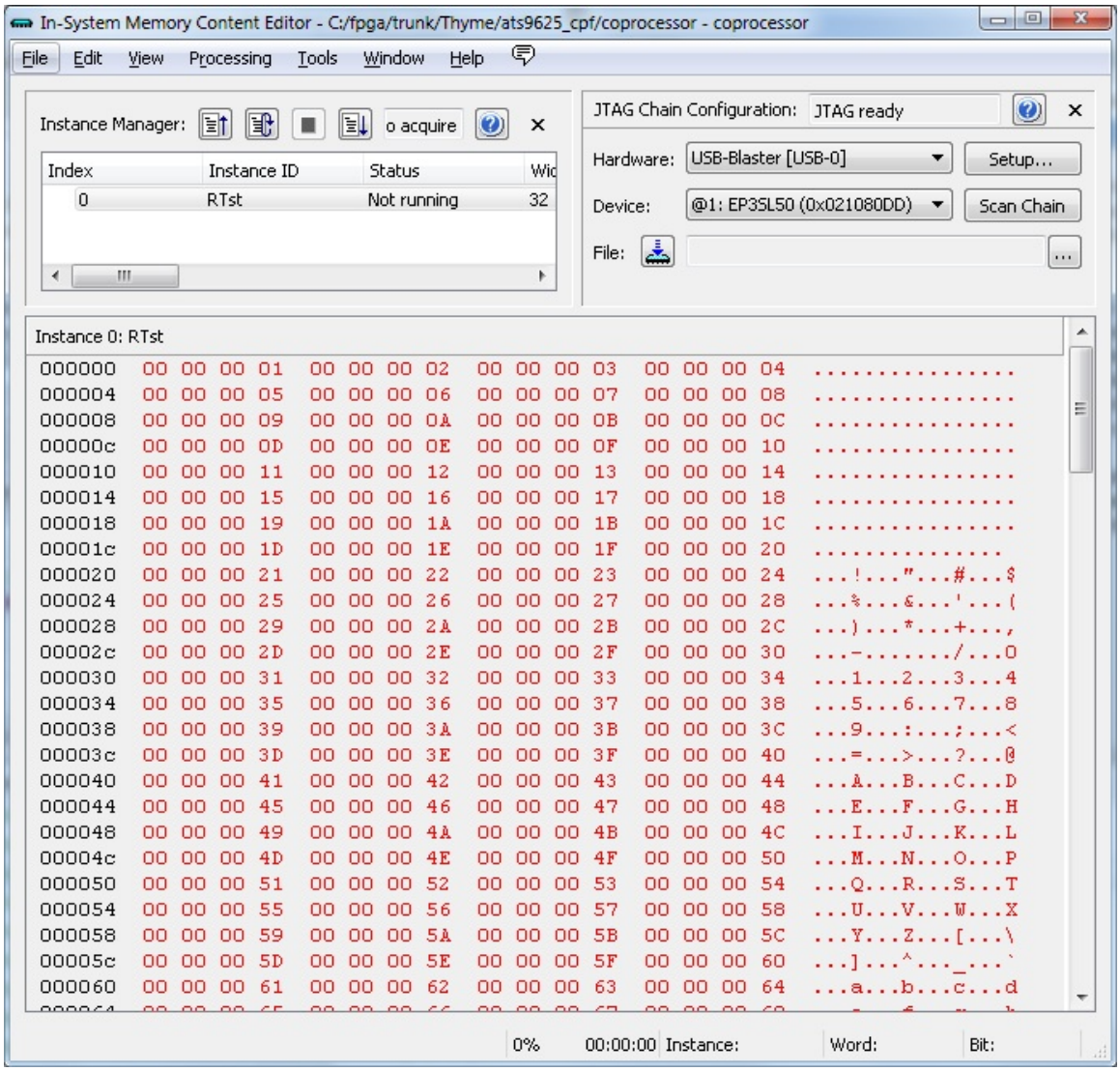


Figure 8-5 In-System Memory Content Editor® Sample Screenshot

Note that the data shown can be hand-modified and written back to the memory. Likewise a .mif/.hex file can be transferred to it. One example use for FPGA ROMs is to change internal wave tables.

Note that this methodology for access to memories is above and beyond the access circuit which a user can create to access internal memories via the Control Bus.

Please refer to the Quartus manual for further information Altera In-System Memory Content Editor®.

© 2003-2012 Alazar Technologies Inc.

9 Simulation Framework

9.1 Overview

A full simulation framework is provided with the FPGA Development Kit, which allows the user get to results very quickly.

Significant effort has been put in to provide the user with a ready-to-use, fully functional simulation framework that can be quickly used out-of-the-box. The simulation framework includes:

- Testbench
- Facilitator Scripts
- Pre-formatted Waveform file
- SlickEdit® workspace and project definition files

The testbench is written in behavioural (non-synthesizable) VHDL to take advantage of the language's power while not requiring expensive additional licenses for more advanced verification techniques (such as assertion-based verification methods). It uses a Directed Testing approach.

Also, since the Altera-specific Modelsim® editions are available directly from Altera at a reduced cost or for free, the framework is targeted to Mentor Graphics' Modelsim® simulator, but can be easily adapted to any other VHDL simulator. The entire testbench is simulator-independent.

The following sections describe each component in detail.

9.1.1 Simulation Folder Hierarchy

The "ats9625_cpf" folder is the simulation working directory, which it also shares with Quartus. Some script files are located in this working directory. However the user-relevant testbench source files are located in `_source_files_user\sim`.

Please refer to section 4.1.3 for details on file and folder locations.

The simulation files located in the main folder are:

modelsim.tcl
simbuild.bat
simrun
wave.do

The files are described below.

© 2003-2012 Alazar Technologies Inc.

9.2 Facilitator Scripts

Even armed with a vendor-supplied testbench, there is always a minimum amount of setup required to get the simulation going. Unless the user has recent experience with the simulation tool and is familiar with the process, this step can be a time-consuming hurdle.

In order to allow the user to start, facilitator scripts are provided. They can be found under the "sim" folder, which is the simulation working directory.

9.2.1 simbuild.bat script

This script takes care of compiling the entire design for simulation. It automatically detects whether this is a first-time run or a subequent run. On a first-time run, it creates the Modelsim "work" library, and optionally maps the Altera libraries if you are using a non-Altera version of Modelsim.

This script is written in the Microsoft Windows®-native Command Prompt / DOS which makes it accessible from almost anywhere and thus integrates easily into the user's working environment. For example, the script can be called up from:

- within Modelsim
- in a Windows Command Prompt window
- in an editor such as SlickEdit® or any other

The scripts sets the appropriate compiler options for both rtl and behavioural logic. Figure 9-1 shows a sample (abbreviated) output from this script:

© 2003-2012 Alazar Technologies Inc.

```
*** No modelsim work library found; creating it now.  This is normal on a first-time run.
*** simbuild.bat script modelsim-altera variable is set to indicate you are using a Modelsim-Altera version.
*** Therefore there is no need to map Altera libraries since they are pre-compiled and pre-mapped for you.
*** If you are not using Modelsim-Altera, you will need to change this variable setting, and precompile
*** Altera libraries per the Quartus instructions.
....
--------------------------------------------------
Compiling coprocessor User design...
--------------------------------------------------

Model Technology ModelSim ALTERA vcom 6.6d Compiler 2010.11 Nov  2 2010
-- Loading package standard
-- Loading package std_logic_1164
-- Loading package textio
-- Loading package numeric_std
-- Loading package utils_conv_pkg
-- Loading package ctl_bus_if_pkg
-- Compiling package user_version_registers_pkg
-- Compiling package user_mem_map_pkg
-- Compiling package body user_mem_map_pkg
-- Loading package user_mem_map_pkg
-- Compiling package user_top_level_pkg
-- Loading package user_mem_map_pkg
-- Loading package user_version_registers_pkg
-- Loading package user_top_level_pkg
-- Compiling entity user_top_level
-- Compiling architecture rtl of user_top_level
....
--------------------------------------------------
Compiling coprocessor testbench...
--------------------------------------------------

Model Technology ModelSim ALTERA vcom 6.6d Compiler 2010.11 Nov  2 2010
-- Loading package standard
-- Loading package std_logic_1164
-- Loading package textio
-- Loading package std_logic_textio
-- Compiling package tb_coprocessor_pkg
-- Compiling package body tb_coprocessor_pkg
-- Loading package tb_coprocessor_pkg
-- Compiling package cpu_mem_map_pkg
-- Loading package cpu_mem_map_pkg
-- Compiling entity cpf_ctl_bus
-- Compiling architecture rtl of cpf_ctl_bus
-- Loading package tb_coprocessor_pkg
-- Compiling entity tb_ctl_bus_master
-- Compiling architecture bhv of tb_ctl_bus_master
-- Loading package numeric_std
-- Loading package math_real
-- Compiling entity tb_adc_data_generator
-- Compiling architecture rtl of tb_adc_data_generator
-- Compiling entity tb_adc_infile_wavgen
-- Compiling architecture bhv of tb_adc_infile_wavgen
-- Compiling entity tb_outfile_data_capture
-- Compiling architecture bhv of tb_outfile_data_capture
-- Loading package utils_conv_pkg
-- Loading package ctl_bus_if_pkg
-- Loading package coprocessor_pkg
-- Compiling entity tb_coprocessor
-- Compiling architecture bhv of tb_coprocessor

--------------------------------------------------
Build complete.  Ready for simulation.
--------------------------------------------------
```

Figure 9-1 simbuild.bat script sample output (abbreviated)

9.2.2 simrun script

Calling up this script from the Modelsim® console window (by typing in "do simrun") will run a complete simulation automatically. It takes care of a number of setup tasks normally required by the user, which greatly facilitates simulation.

This Modelsim®-native "dofile" script performs the following tasks:

- Set up the transcript log file
- Set appropriate Modelsim runtime defaults
- Search all subdirectories for *.hex and *.mif files, and copy them over to the working directory. This solves the often misunderstood problem of files not found at simulation runtime, and avoids tedious and error-prone manual copying.
- Call up the appropriate wavefile
- Log all signals so they can be pulled into the Wave window without the need to restart the simulation.
- Monitor the Testbench to end the simulation automatically at the desired time.
- Set the default view to the waveform window (or transcript window, optionally)
- Provisions to handle multiple testcases with separate wavefiles.

Built-in support is provided for multiple testcases, each with their own wavefiles, should the user decide to have multiple testcases and break them out of the testbench.vhd file. See in-file comments for details.

9.2.3 modelsim.tcl

This script is automatically called up by Modelsim® when it starts up. Putting runtime option customizations in this file allows the user to save modelsim preferences independently of Modelsim installs. Refer to the Modelsim® user guide for more details.

9.3 Wavefile (wave.do)

The pre-formatted wavefile contains all the relevant signals for the user to look at, cleanly organized within functional groups at different hierarchy levels. This file in itself is sufficient for simple applications, and can be easily expanded to include more user design specific signals.

It provides at-a-glance view of the input and output signals from the FPGA as well as from the user_top_level block, including Datapath, Trigger and Auxiliary signals, debug LEDs, and Control Bus.

The wavefile also make use of the Analog waveform display capabilities of Modelsim®, to provide an intuitive time-domain comparative view of the input and output data, on both channels.

See below for sample screenshots.

© 2003-2012 Alazar Technologies Inc.

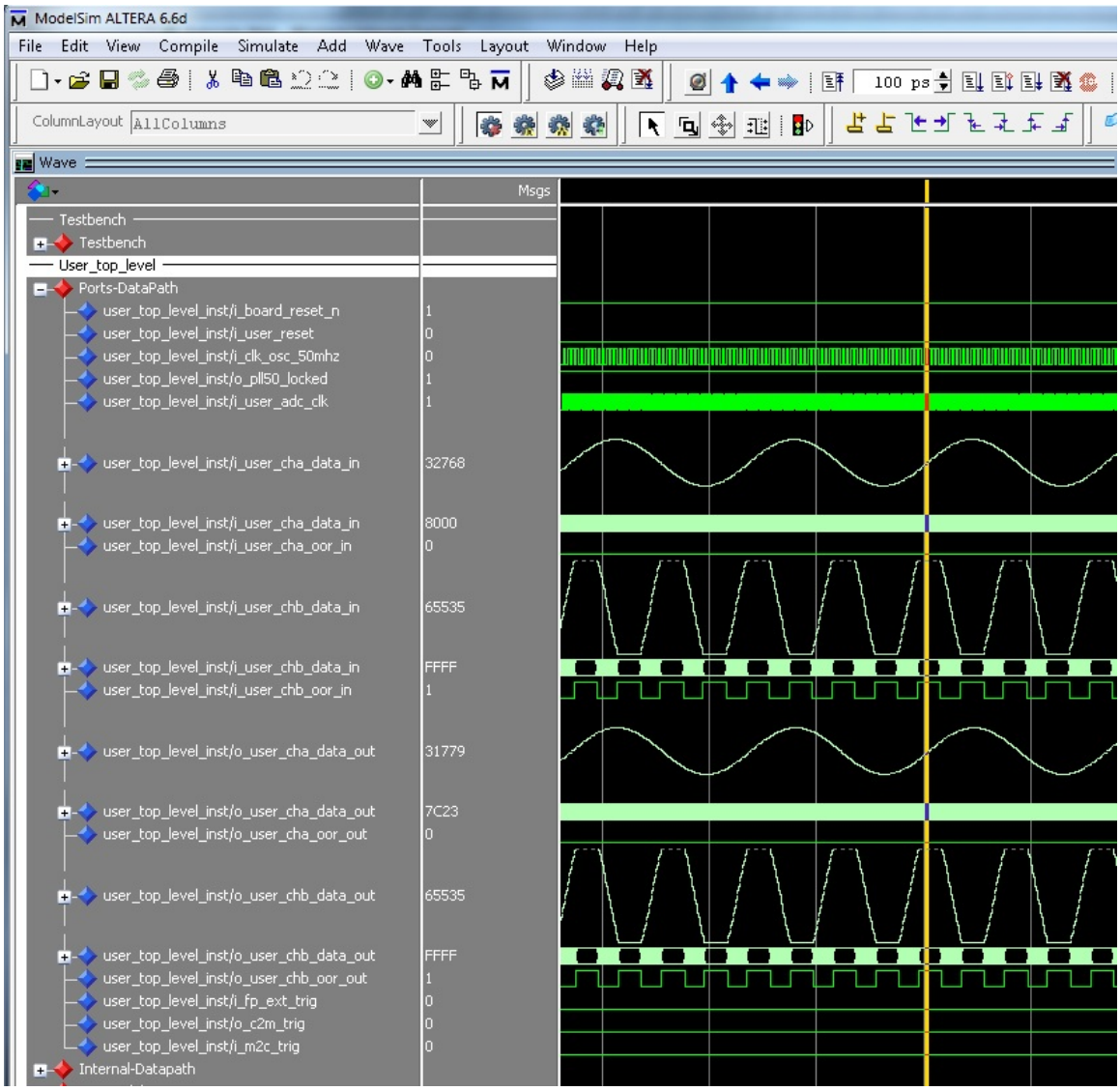


Figure 9-2 Wavefile - Datapath sample

Figure 9-2 demonstrates a clean sine wave on channel A, while channel B has a signal that goes Out-Of-Range, demonstrating the "OOR" signal behaviours.

© 2003-2012 Alazar Technologies Inc.

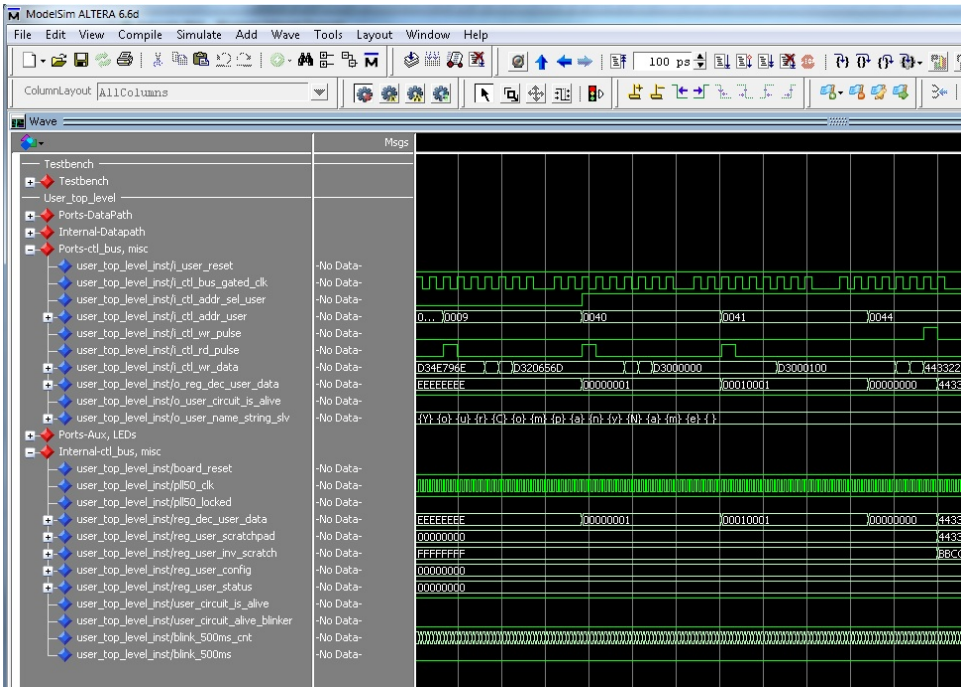


Figure 9-3 Wavefile - Control Bus sample

Figure 9-3 demonstrates Control bus read and write transactions in both Alazar and the User register spaces.

9.4 SlickEdit® Workspace and Project Files

SlickEdit® is a completely optional 3rd-party editor and integrated development environment. Sample workspace and project definition files are provided should the user also happen to be using this same application. The project file includes all source files, as well as the appropriate compile settings and options to allow compiling, building and simulating at a keystroke.

The files can be found at the root directory (ats9625):

- ats9625_cpf.vpw
- ats9625_cpf.vpj

Please note that these are provided for user convenience only, on an as-is basis with no support. Purchasing information can be found at http://www.slickedit.com

9.5 Testbench

The FPGA Development Kit includes a fully-featured VHDL Testbench which is intended as a framework to use as a base platform, so the user can simply modify or add components as required to suit new custom functions.

It is pre-configured to exercise and test the provided circuit features.

© 2003-2012 Alazar Technologies Inc.

9.5.1 Features

The self-contained testbench features the following:

- Self-verifying support with pass/fail report
- Simple sine wave input generator, to quickly provide various frequency inputs to user functions
- Input waveform generator which reads data samples from a user-defined input text file
- Excel spreadsheet also provided to generate above input text files quickly
- Output capture-to-file, with raw data in native format (unsigned binary) and built-in conversion to signed integer format.
- Control bus access functions (read, write, read-verify)
- User abstraction from the DDR datapath formats
- Full access to all signals (Channel A & B inputs and outputs, Trigger, Auxiliary, LED debug signals)

9.5.2 File and Module Descriptions

All user-relevant testbench files are located under the ats9625_cpf_source_files_alazar\sim folder. Here is a list of the testbench files and their descriptions:

tb_coprocessor.vhd: This is the top-level testbench file, which also contains the testcase.

tb_coprocessor_pkg.vhd: The package file used by tb_coprocessor.vhd. Defines global-scope variables for the control bus accesses, the write, read and read-verify procedures, and component declarations.

tb_adc_data_generator.vhd: A basic sine wave Data Generator. See section 9.5.2.1 for more information.

tb_adc_infile_wavgen.vhd: An Arbitrary Waveform Data Generator. See section 9.5.2.2 for more information.

tb_outfile_data_capture.vhd: Waveform capture-to-file module. See section 9.5.2.4 for more information.

data_in_adc_a/b.txt: input text file for the default Channel A/B Arbitrary Waveform Data Generator. See section 9.5.2.2 for more information.

data_out_cha/chb.txt: output text file from the default Channel A/B Waveform capture-to-file module. See section 9.5.2.4 for more information. These files are produced and overwritten each time the simulation is executed.

© 2003-2012 Alazar Technologies Inc.

waveform_sample_generator.xls: This is a sample Microsoft Excel® spreadsheet that can be used to generate input sine wave data for the Arbitration Waveform Data Generator. See section 9.5.2.3 for more information.

The ats9625_cpf_source_files_alazar\sim folder contains a few additional files which should not be modified by the user.

9.5.2.1 Basic Sine Wave Generator

(tb_adc_data_generator.vhd)
This is an ADC model which generates a simple sine wave, with amplitude and frequency inputs. It allows the user to quickly generate an input signal without having to write any code.

9.5.2.2 Arbitrary Waveform Data Generator Module

(tb_adc_infile_wavgen.vhd)
This is an ADC model which generates an arbitrary waveform file by reading samples from a user-defined input file. It allows the user to generate input signal using 3rd party tools, such as Matlab, live data capture, Excel, Text Editor, etc.

It reads a text input file with 1 sample value per line, and "plays back" the samples to generate a waveform, at one sample per clock. The data is expected to be integer signed (e.g. 205, 0, -205) scaled to the ADC's dynamic range. The entire range is 16 bits (-32768 to +32767). The waveform is repeated infinitely; once the last sample is read from the file, it starts over.

NOTE: The last data line MUST have a CARRIAGE-RETURN at the end of the sample value, else it will be IGNORED.

The module has a parameterizable file name, can be enabled/disabled, and in addition to the native ADC DDR 8-bit output, it also provides a single-data-rate 16-bit output, for user readability in the simulation wave window.

When the enable is low, it will still provide an active DDR output which correctly represents the ADC zero-value. When in reset the bus will be set to zero, representing the ADC in reset. When out of reset but disabled, it sends the ADC's zero-value (8000h).

9.5.2.3 Waveform Sample Generator Spreadsheet

(waveform_sample_generator.xls)
This is a sample Microsoft Excel® spreadsheet that can be used to generate input sine wave data for the Arbitration Waveform Data Generator. Comments and instructions are embedded in the file.

© 2003-2012 Alazar Technologies Inc.

The user can enter the desired frequency, then a set of samples is produced, which can be copied and pasted into a text file to quickly generate a dataset for the Arbitrary Waveform Data Generator Module.

9.5.2.4 Data capture-to-file Module

(tb_outfile_data_capture.vhd)
This module captures data to a text file. It captures the raw data in native format (the ADC's unsigned binary) and features built-in conversion to signed integer format. Both formats are output to the file, as well as the OOR (Out-Of-Range) Indicator to qualify the data.

The file is output with space-separated values than can easily parsed by a script or be imported into a spreadsheet application such as Microsoft Excel® and manipulated from there.

This module has a parameterizable file name, features an enable signal so the testcase can dictate when to start and stop capturing data.

Note that it is modular and generic in nature and can be used to capture data to file at any other convenient location in the design.

9.5.3 Testbench Code Description

The testbench top-level is tb_coprocessor.vhd. It instantiates the entire Coprocessor FPGA (CPF) as a DUT (Device Under Test), and pulls in other generator and analyzer modules to simulate the board devices around the CPF.

9.5.3.1 Testcase Master Process

The testbench includes a time-sequential testcase inside a single process labelled "tc", which contains series of signal changes on the board and to the generator/analyzer modules, as well as procedure calls; all this to generate the desired stimuli and analyze the results as required.

The tc process controls the flow of time in the simulation. Procedures that also expend time can be written and called up by the testcase (such as the supplied cpu_read/write procedures), and concurrent processes can be controlled by the testcase process' signals, to control generators and analyzers for example.

The testcase can and should be modified by the user to generate the desired stimuli.

9.5.3.2 Self-Monitored Tests and Testcase Pass / Fail Reporting

The testbench provides the facilities for self-monitoring and Pass/Fail reporting. The tc_fail_cnt integer signal can used in the testcase as well as passed on to test procedures. Test failures increment the value, thus allowing the testcase to tally up test failures.

© 2003-2012 Alazar Technologies Inc.

At the end of the testcase, if tc_fail_cnt is 0 then the self-monitored tests all passed, and a messages announces the good news. If it is non-zero then there is a corresponding failure message to inform the user of how many self-monitored tests failed.

As mentioned in the simulation report, the pass/fail report only applies to the tests in the testcase that are self-monitoring. Many tests may not be, and therefore will require the user to analyze the signal waveforms visually and determine correctness. Implementing self-monitoring tests requires more up-front work and maintenance, but provides more efficient coverage and time savings in regression testing. It is up to the user to decide which method is most efficient to implement for their application. The framework is built to allow a hybrid method, supporting both simultaneously.
The following figures demonstrate passing and failing tests, respectively:

```
# Read from user section on CTL bus.
# Verifying signature.
# Read Addr:       0x0040   Data: 0x00000001      Read verified ok.
# Reading out Major and Minor Versions.
# Read Addr:       0x0041   Data: 0x00010001
# Write in User section.  Scratchpad Register.
# Writing Addr:    0x0044   Data: 0x44332211
# Read from cpf on CTL bus.
# Read Addr:       0x0044   Data: 0x44332211      Read verified ok.
# Read Addr:       0x0045   Data: 0xBBCCDDEE      Read verified ok.
# Front Panel Trigger propagation.
#
# Starting ADC data Testing.
# Turning on ADC data.
# tb_outfile_data_capture.vhd:  Starting Data Capture into file: data_out_chb.txt
# tb_outfile_data_capture.vhd:  Starting Data Capture into file: data_out_cha.txt
#
# Simulation Ending Normally.
# ************************************************************
# ********        Tests Passed.  Congratulations!        *********
# ************************************************************
# Note: Only specific tests are evaluated for Pass/Fail.
# Simulation stop requested.
```

Figure 9-4 Sample Transcript of all tests passed

© 2003-2012 Alazar Technologies Inc.

```
# Read from user section on CTL bus.
# Verifying signature.
# Read Addr:       0x0040   Data: 0x00000001      Read verified ok.
# Reading out Major and Minor Versions.
# Read Addr:       0x0041   Data: 0x00010001
# Write in User section.  Scratchpad Register.
# Writing Addr:    0x0044   Data: 0x44332211
# Read from cpf on CTL bus.
# Read Addr:       0x0044   Data: 0x44332211      *** Read does not match expected value!
#   Read Value:      44332211
#   Expected Value:  00000000
#   Mask:            FFFFFFFF
# ** Warning: **** Read value does not match expected value.
#    Time: 2760 ns  Iteration: 4  Instance: /tb_coprocessor
# Read Addr:       0x0045   Data: 0xBBCCDDEE      *** Read does not match expected value!
#   Read Value:      BBCCDDEE
#   Expected Value:  FFFFFFFF
#   Mask:            FFFFFFFF
# ** Warning: **** Read value does not match expected value.
#    Time: 2920 ns  Iteration: 4  Instance: /tb_coprocessor
# Front Panel Trigger propagation.
#
# Starting ADC data Testing.
# Turning on ADC data.
# tb_outfile_data_capture.vhd:  Starting Data Capture into file: data_out_chb.txt
# tb_outfile_data_capture.vhd:  Starting Data Capture into file: data_out_cha.txt
#
# Simulation Ending Normally.
# ***************************************************************
# ********              TESTS FAILED                     *********
# ***************************************************************
# ** Number of Failures: 2
# ** Refer to simulation log for more detail as the failures occurred.
# Simulation stop requested.
```

Figure 9-5 Sample Transcript of a few failing tests

9.5.3.3 Control Bus Accesses

The testbench provides a framework for the user to access the control bus just as the host CPU would in software. This provides a form of "abstraction layer" so the user does not have to worry about the detailed signal manipulations required to perform control bus transactions.

This consists of three (3) procedures, a set of global signals and a cpu transactor process.

The 3 access procedures can be used directly from the testcase. This is done in the example testcase and can be copied and adapted as needed. The procedures are:

- cpu_write (cpu_ctrl, address, input data[31:0], verbosity)
- cpu_read (cpu_ctrl, address, output data[31:0], verbosity)
- cpu_read_verify (cpu_ctrl, address, output data[31:0], tc_fail_cnt, verify_bit_mask, verbosity)

The cpu_ctrl is a global signal record needed by the procedure to communicate with the cpu transactor process which actually generates the signals for the transaction.

The address is of type "natural", which makes it easy to use and most importantly allows the use of the address index constants defined in the user memory map package. Thus

the user can reassign addresses in the memory map without needing to modify the testbench code to match. This concept is demonstrated in the testbench as it is.

9.5.3.4 Verbosity Control

The tb_coprocessor_pkg.vhd defines a boolean constant called "default_verbosity". This is used currently by the ctl bus access procedures, but can be used throughout the testbench.

The constant can be used by various procedures, processes and the testcase to provide or suppress output messages to the simulation transcript window.

Refer to the cpu_read and write procedures in tb_coprocessor_pkg.vhd for examples on how to use it.

9.5.3.5 Stimulus Generators

The are various stimulus generators in the testbench to generate the appropriate clocks and aligned signals.

9.5.3.6 Sample Analyzer: Trigger Monitor

A sample analyzer process is included in the testbench, which monitors the propagation of the Front Panel External Trigger to the Coprocessor-to-Main FPGA Trigger signal. It raises a simulation warning if the signal does not propagate in a certain amount of time.

9.5.3.7 c2m_data_channel_separator process

The CPF's output datapath combines Channel A and Channel B into a single DDR bus. Thus it is tedious and ineffective to visually unravel the data every cycle when viewing the output datapath directly in the simulator wave window.

This process splits out the output datapath into two SDR buses for intuitive viewing in the simulator wave window, allows analog wave viewing, and also allows post-processing and analysis on the output data (such as output to file for example).

9.5.3.8 Optional Expansion to Multiple Testcases

While the current testbench embeds a single testcase, the framework is built such that a user could extract the testcase and place it hierarchically above the testbench. This would allow multiple testcases to be written, each testing their own features, and each having their own custom wave files.

The user desiring to do this must customize the testbench, bring out the required ports as needed to implement this.

© 2003-2012 Alazar Technologies Inc.

The "simrun" script has provisions to support this (with basic modifications). It can accept a parameter, the testcase's name, then call up the right testcase and use the appropriate wavefile.

This is provided for the user's convenience and without support, and the detailed modifications required are outside the scope of this document.

© 2003-2012 Alazar Technologies Inc.

Book II - SingleClk Framework with FIR & DDC Example

-- BOOK II --

FDK SingleClk Framework with FIR & DDC Example

© 2003-2012 Alazar Technologies Inc.

Book II Contents

10 Book II - Introduction (SingleClk FIR & DDC) 79
11 Functional Description 80
11.1 ATS962x Waveform Digitizer Board Block Diagram 80
11.2 Top-Level Block Diagram (DDC Design) 81
11.3 Functional Overview 81
11.3.1 High-Level Functionality 81
11.3.2 Data Flow and Sample rates 82
11.3.3 Output Selector 82
12 Detailed Functional Description 84
12.1 NCO 84
12.1.1 NCO Specifications 84
12.1.2 Post-multiply Re-scale 84
12.2 FIR-only (no-Demodulation) Option 84
12.3 Filtering & Decimation 84
12.3.1 Post-FIR Binary 16-bit-slice Rescaler 85
12.3.2 Changing FIR Coefficients 85
12.3.2.1 Generating and Scaling FIR Coefficients 86
12.4 Output Selection and Adaptation 86
12.4.1 Output Format 87
12.5 Acquisition Manager 87
12.5.1 Free-Running Operation 87
12.5.2 Trigger-Controlled Operation 87
12.6 Buffer Header (optional function) 87
12.6.1 Buffer Header support limitations 88
12.6.2 Buffer size settings 88
12.7 GPS Timestamp (optional function) 88
12.7.1 GPS Buffer Timestamp 88
12.7.2 One-second Frequency Counter 89
13 Memory Map 90
13.1 Memory Map - Alazar Section (Reserved) 91
13.1.1 cpf_version [Addr 1] 91
13.1.2 cpf_variant [Addr 2] 91
13.1.3 compilation_timestamp [Addr 3] 92
13.1.4 cpf_config [Addr 5] 92
13.1.5 cpf_status [Addr 6] 92
13.2 Memory Map - User Section, Alazar-Defined 93
13.2.1 user_version [Addr 33] 93
13.2.2 data_format_cha/chb [Addr 36, 37] 94
13.2.3 bytes_per_buffer_out 94
13.3 Memory Map - User Section, Open (Customizable) 95
13.3.1 user_cfg [Addr 128] 96
13.3.2 user_status [Addr 129] 98
13.3.3 latch_regs [Addr 130] 98
13.3.3.1 About latched registers 99

© 2003-2012 Alazar Technologies Inc.

13.3.4 nco_phase_incr [Addr 131] 99
13.3.5 user_acq_cfg [Addr 132] 99
13.3.6 acq_mgr_byte_cntr [Addr 133] (Latched Register) 100
13.3.7 acq_mgr_word_cntr [Addr 134] (Latched Register) 100
13.3.8 buf_seq_num_cntr [Addr 135] (Latched Register) 100
13.3.9 fir_coeff_cfg [Addr 140] 100
13.3.9.1 FIR Coefficient Read/Write Accesses 101
13.3.10 fir_coeff_addr [Addr 141] 101
13.3.11 fir_coeff_in_data [Addr 142] 101
13.3.12 fir_coeff_out_data(4..1) [Addr 143-146] 102
13.3.13 rescaler_slice_pos [Addr 147] 102
13.3.14 gps_freq_cnt [Addr 148] (latched register) 102
14 Special Considerations 103
14.1 Acquisition Mode 103
14.2 Data latency 103
14.2.1 Data latency vs Trigger 103
14.2.2 Data latency vs Timestamp 103
14.2.2.1 Consequence for Trigger-Controlled operation 103
15 Quick Start Guide 104
15.1 Downloading the FIR/DDC FPGA 105
15.2 Read/Write Accesses to CPF FPGA 105
15.3 Settings: DDC & FIR operation 105
15.4 Settings: FIR-only operation 106
15.5 Settings: Header Insertion Enable 106
15.6 Settings: Transparent 107
15.7 Special Notes when using AlazarDSO with FIR/DDC FPGA 107
15.7.1 AlazarDSO Not Decimation-aware 107
15.7.2 Channels 1&2 must always be enabled 108
15.7.3 Sample screenshot 108

© 2003-2012 Alazar Technologies Inc.

List of Figures

Figure 11-1 ATS9625 Board Block Diagram 80
Figure 11-2 FPGA Block Diagram (DDC Sample Design) 81
Figure 15-1 AlazarDSO Oscilloscope view, with Header Insertion 107
Figure 15-2 AlazarDSO Spectrum Analyzer view - Demodulated & Filtered data 108

List of Tables

Table 12-1 Buffer Header Contents 87
Table 13-1 Memory Map - Global 90
Table 13-2 Memory Map - Alazar Reserved Section 91
Table 13-3 Memory Map - User Customizable Section 93
Table 13-4 Memory Map - User Customizable Section 95
Table 13-5 Summary of user_cfg Settings and Output Data Format 98
Table 15-1 Quick Start Configuration Summary 104
Table 15-2 FPGA Settings Summary 104

© 2003-2012 Alazar Technologies Inc.

10 Book II - Introduction (SingleClk FIR & DDC)

The FIR and DDC (Finite-Impulse Response Filter, and Digital Down-Converter) example design is a fully featured FPGA data processor, useable out-of-the-box for some practical real-world applications. It is easily customizable to suit various requirements.

It is based on the Single Clock Framework.

© 2003-2012 Alazar Technologies Inc.

11 Functional Description

This section provides technical details on the functionality and parameters for each of the functional blocks, starting with block diagrams.

11.1 ATS962x Waveform Digitizer Board Block Diagram

Please refer to the ATS962x Coprocessor FDK-Designer's Guide for more details. The following figure is repeated here for convenience. The ATS9625 has AC-Coupled signal inputs, while the ATS9626 has DC-coupled inputs.

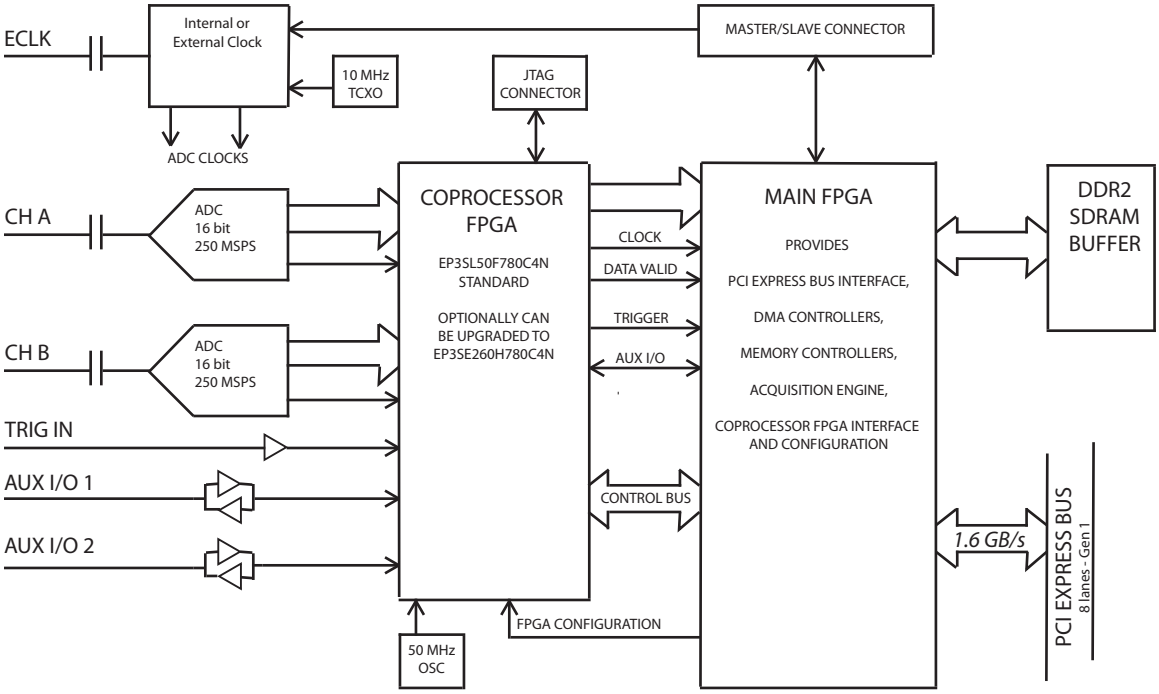


Figure 11-1 ATS9625 Board Block Diagram

11.2 Top-Level Block Diagram (DDC Design)

The following figure provides a detailed visual overview of the example DDC design, which is located in the Coprocessor FPGA, as seen in Figure 4-1.

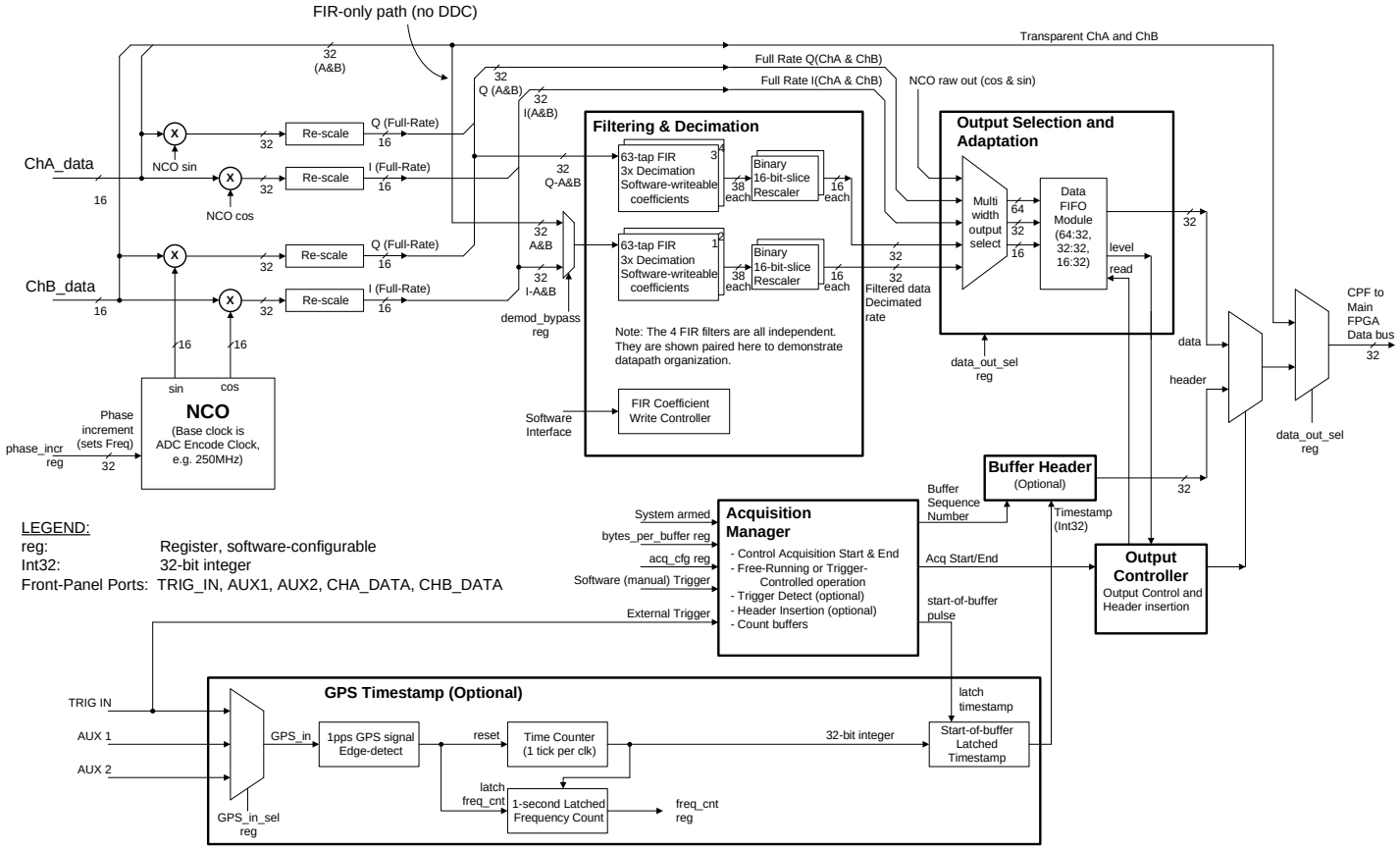


Figure 11-2 FPGA Block Diagram (DDC Sample Design)

11.3 Functional Overview

11.3.1 High-Level Functionality

This sample design can be used for the following high-level functionality:

- Digital Down-Converter. Data from 1 or 2 channels is demodulated by a programmable NCO frequency. Various output options are provided to limit the data throughput as desired.
- Simple decimating FIR filter application. The data from both ADC channels is filtered and decimated. (no demodulation)
- Transparent data acquisition. The ADC's raw Channel A and Channel B data is provided as is.
- Note that the design provides limited triggering functionality.

© 2003-2012 Alazar Technologies Inc.

An optional header containing sequence number and GPS timestamp information is added to the output data. The header is not available in some of the non-decimated data paths.

11.3.2 Data Flow and Sample rates

Referring to the board block diagram of Figure 4-1. Data from both channels is sampled by the onboard ADC with 16-bit precision, at a nominal rate of 250MS/s. The sampling rate can be changed using the onboard clock synthesizer, for rates in the range of 50MHz to 250MHz, in 5MHz increments. Alternatively, an External clock can be supplied. Please refer to the ATS962x Waveform Digitizer User Manual for details. Regardless of the clock source and frequency, both ADCs run at the same rate, and the ADC data and clock are provided to the FPGA. Since this example is based off the SingleClk reference design, the FPGA's entire datapath runs at the same frequency as the ADC's.

Referring to the top-level block diagram of Figure 17-2, the following describes internal Coprocessor FPGA datapath functionality:

The NCO (Numerically-Controlled Oscillator) has a software-configurable phase increment, and thus the user can set the desired demodulating frequency.

Channel A and Channel B data is multiplied by the NCO's cosine and sine outputs to generate in-phase (I) and quadrature (Q) demodulated signals, respectively. The signal is re-scaled back to the input range after the multipliers.

The input to the Filtering and Decimation block is configurable such that the following can be filtered and decimated:

- Channel A and Channel B raw data from ADC. This allows the use of the FIR filters without any demodulation.
- I&Q demodulated data from Channel A
- I&Q demodulated data from Channel A and I&Q demodulated data from Channel B.

The Filtering and Decimation block consists of 4 separate but identical FIR (Finite Impulse Response) decimating filters. They have a fixed decimation factor. The FIR filters have software-configurable coefficients which can be changed at any time. Note that filtered data will be affected during the process however, as coefficients get loaded.

The Data FIFO module takes care of organizing the data for the output bus.

The Acquisition Manager offers a few modes of operation (Free-running, Trigger-Controlled), as well as optional insertion of information (sequence number, timestamp) as a header in the output data stream.

11.3.3 Output Selector

The output of the Coprocessor FPGA is software-selectable, to any one of the following:

© 2003-2012 Alazar Technologies Inc.

- Transparent channel A and Channel B data (full rate, unprocessed)
- Full-Rate I demodulated data from Channel A & B
- Full-Rate I&Q demodulated data from Channel A
- Full-Rate I&Q demodulated data from Channel B
- Decimated and filtered I demodulated data from Channel A and/or Channel B.
- Decimated and filtered I&Q demodulated data from Channel A and/or Channel B.
- Decimated and filtered raw ADC data from Channel A and/or Channel B (not demodulated. This is a FIR-filter operating mode.)

Note that it is not possible to obtain the Full rate I&Q for both channels, as the system does not have enough bus bandwidth to support these effective 4 channels of data. This would be double the bandwidth of the standard 2-channel ADC operation.

© 2003-2012 Alazar Technologies Inc.

12 Detailed Functional Description

12.1 NCO

The NCO is a multiplier-based NCO running at the ADC clock. Its phase increment is software-configurable through a register. Through this the user is able to generate any desired frequency. The frequency can be changed at any time during operation.

Its default is set to:

- 70MHz (assuming an ADC clock frequency of 250MHz).

12.1.1 NCO Specifications

The NCO is an Altera Megacore and is fully described by Altera documentation. Here are some details of note, for convenience.
The NCO is implemented using the Multiplier-Based option, with a Phase Accumulator precision of 32 bits, Angular precision of 16 bits and 16-bit Magnitude. All configuration details are accessible by using Altera's Quartus Megawizard tool, and editing the nco source file.

Please refer to the nco_phase_incr register for more details.

12.1.2 Post-multiply Re-scale

Since the NCO's 16-bit output is multiplied by the 16-bit sample input, this results in a 32-bit value. This value is rescaled down to a 16-bit value, such that the original ADC dynamic range is maintained. The rescale value assumes full-range NCO output and is thus set to divide by 0x8000.

12.2 FIR-only (no-Demodulation) Option

If the desired functionality is FIR filtering and decimation only, without demodulation, the user can bypass the NCO and multipliers and feed the raw ADC's Channel A and Channel B data directly into the Filtering & Decimation block.

This is done via the no_demod_sel register, which controls the multiplexer at the input of the the Filtering & Decimation block.

12.3 Filtering & Decimation

Please refer to section 11.3 Functional Overview.

The FIR parameters are as follows:

- x3 decimation (output rate is 1/3 of the input rate). Cannot be changed in software; can be changed in FPGA design.
- 16-bit coefficients, signed binary format (16-bit signed integer)

© 2003-2012 Alazar Technologies Inc.

- 63-tap with independent coefficients. There is no assumption of coefficient symmetry; all taps are physically separate and take coefficients independently.
- 40MHz default coefficients set to low-pass filter, 3db cutoff at 40MHz assuming a 250MHz sampling frequency. Coefficients scaled using power-of-2 multiple so near-unity gain can be achieved using the Post-FIR Binary rescaler.

The 4 filters' coefficients are independent from each other, and are user-programmable via software. Please refer to the memory map section.

The decimation factor can be changed through the Altera Quartus Megawizard FPGA software, and require the compilation and generation of a new FPGA file. No VHDL source code change should be needed however.

The number of taps, coefficient format and precision can also be changed, but depending on the selected choices, may require some VHDL source code changes additionally to the above.

12.3.1 Post-FIR Binary 16-bit-slice Rescaler

The FIR filter scales the signal up by more than double the number of input bits. It must therefore be rescaled down to 16 bits per sample. The post-FIR bitslice rescaler is software configurable to take a specified 16-bit slice of consecutive bits within the FIR output word. This therefore gives a power-of-2-multiple division factor.

If the user wants 1:1 scaling from system input to output, then it is suggested that the FIR coefficients be scaled such that the FIR scaling is a power of 2 multiple, and therefore can be divided back down for unity gain by the bitslice rescaler.

Conversely, for maximum dynamic range, the user can scale the FIR coefficients to fully utilize the FIR output's range, however the post-FIR scaling will result in a non 1:1 input to output scaling through the system.

12.3.2 Changing FIR Coefficients

The FIR coefficients can be changed to achieve the desired cutoff frequency and gain. There are mainly 2 ways of changing the default FIR coefficients, one via software, the other by changing and recompiling the FPGA.

The first method is to update the coefficients via software, by writing to the fir_coeff registers as described in the memory map section. This method does not require recompiling the FPGA, but one must program the coefficients everytime the FPGA is downloaded or the board restarted. It is also the only way to have a different set of coefficients for the 4 FIR filters.

The second method requires the use of Altera-Quartus, and requires recompiling the FPGA. This is done by starting Quartus' Megawizard tool, and editing the "fir_main.vhd" entity. New coefficients can be imported into the Megawizard tool and the FIR re-generated. If the user only wants to change coefficients and nothing else, be

© 2003-2012 Alazar Technologies Inc.

careful to only specify a single set of precisely 63 coefficients, otherwise VHDL design changes may also be required. For example a higher number of coefficients could result in a larger width in output bits, etc.

12.3.2.1 Generating and Scaling FIR Coefficients

In most instances, users will have access to DSP tools such as Matlab to generate and scale coefficients as desired for their application.

Here are steps which can be used to generate new coefficients by using Quartus only. This is provided for convenience only for knowledgeable users of Quartus, and is not supported by Alazar.

This design uses the FIR II core from Altera, which requires that user import a set of coefficients, but it cannot generate them. However the FIR I core megawizard has a built-in tool to generate coefficients. These can be generated and exported, modified using a custom .tcl script, then imported into the FIR II core.

1. Edit the "_source_files_user\ip_altera\fir_1_for_coeff_generation\ fir_1_for_coeff_generation.vhd" IP using the megawizard. In the megawizard, enter the desired cutoff frequency and operating frequency.
2. You can keep the current scaling option, which is "Auto with Power 2", which allows unity gain combined with the design's post FIR binary rescaler. Or you can select Auto, which maximizes SNR.
3. Generate this IP so the coefficients output file is created.
4. Open a Command Prompt in the "fir_1_for_coeff_generation" IP's directory.
5. Type the following command in: `quartus_sh -t join_cr_into_commas.tcl`
6. This generates an output file of the coefficients, in the correct format for the FIR II core. It is called "fir2_coef_from_fir1.txt"
7. Edit the "fir_main" IP using the megawizard. Browse for the above-mentioned file and Apply. This will now apply the new coefficients. Verify the list of coefficients to confirm the change.
8. Click Finish to generate the new FIR II core. The change is always applied to all 4 FIR instances, since they are 4 identical instances of the same source.
9. Compile the design, and use its .rbf from now on.

12.4 Output Selection and Adaptation

This module first handles the selection of the desired data lanes according to a user register. Depending on the number of channels and/or I&Q selection, the datapath can be 16, 32, or 64 bits wide. The internal Data FIFO Module then takes care of adapting these input widths to an output of 32 bits. It also stores the data that continues to flow in while the header is being inserted to the output buffer.

The module thus generates 32-bit wide output data by either packing 16-bit data together or spreading 64-bit data across 2 words. Please refer to the user_cfg register description for details on the output selection.

© 2003-2012 Alazar Technologies Inc.

12.4.1 Output Format

The output data format for each sample is 16-bit unsigned binary offset at 8000h. In other words, the zero value is at midrange (8000 hex). This is the same format as the ADC provides to the FPGA.

Note that internally to the FPGA, the data is converted to 16-bit signed binary (two's complement) at the input and output edges, such that all DSP operations are performed in signed binary.

12.5 Acquisition Manager

The acquisition manager controls the acquisition and provides various parameters, options and modes of operation.

12.5.1 Free-Running Operation

This mode of operation is the simplest. Acquisition begins when software dictates it, and data streams continuously until software ends it. The Trigger signal is ignored.

12.5.2 Trigger-Controlled Operation

Trigger-Controlled operation is the same as Free-Running, except that when software starts the acquisition, no data is acquired until the first occurrence of a Trigger. Once acquisition has begun from that first trigger, further Triggers are ignored until the acquisition is stopped and started again.

The Trigger is normally an external signal connected to the TRIG IN front-panel input. Note that software can also force a trigger via a register.

In the case where the GPS Timestamp option is used, the trigger can be the GPS 1pps signal, or it can be a separate signal on a different front-panel connector.

12.6 Buffer Header (optional function)

The buffer header is an optional function whereby a 32 byte header is pre-pended to the data buffer and contains the following information:

Dword Number	Description
1	Header ID & version (5A5A 0100h)
2	Buffer Sequence Number (since start of acquisition, starts at 1)
3	Timestamp (32-bit unsigned int; counter value at start of buffer)
4	Buffer size (Register contents of reg_bytes_per_buffer_out)
5	reg_user_cfg: current register's contents
6	reg_user_acq_cfg: current register's contents
7	<reserved> (0000 0000h)
8	<reserved> (0000 0000h)

Table 12-1 Buffer Header Contents

© 2003-2012 Alazar Technologies Inc.

The function is enabled/disabled via a software register in the FPGA, please refer to the Memory Map section.

12.6.1 Buffer Header support limitations

WARNING: The header must not be enabled on full-rate data.
The header adds data to the stream, and the FPGA's maximum output bandwidth is equivalent to 2 channels at full-rate (1 sample/clk).
Header insertion requires decimation (from the FIR filters, in this case a reduction x3) in order to have bandwidth to insert the header information.
Enabling header insertion in non-FIR output modes will result in data corruption due to internal FIFO overflows.

12.6.2 Buffer size settings

With header enabled, the <bytes_per_buffer_out> register value is used to dictate how much sample data comes after each header.

The data stream therefore consists in the following, repeatedly:

- 32-byte header
- <bytes_per_buffer_out> bytes of data.
- 32-byte header
- <bytes_per_buffer_out> bytes of data.
- ... and so on

Thus, from a system software perspective, the system DMA buffer size (not to be confused with the FPGA's buffer size), should be of size:

n x (<bytes_per_buffer_out> + 32 bytes)

where n in an integer n >= 1.

The user may want to set the system DMA buffer size to a multiple of this size (i.e. n>1), for any of multiple reasons, such as PCI Express transfer performance, software buffer management etc.

The system DMA buffer size is set in software when configuring the acquisition, whereas the FPGA's buffer size is set in the CPF FPGA's `bytes_per_buffer_out` register.

12.7 GPS Timestamp (optional function)

12.7.1 GPS Buffer Timestamp

The GPS Timestamp function counts time continuously at the rate of 1 increment per ADC clock cycle (typically 250MHz which means 4.0ns). If header insertion is enabled, the current time value is placed in a header at the start of each new data buffer. User

software can then use that information as desired. The only thing that changes this counter is the the GPS input, typically a 1 pulse-per-second (pps) signal, which resets the counter everytime an edge of the chosen polarity occurs. This can be seen in Figure 17-2.

The user's 1 pps GPS_in signal can be connected into any of 3 input ports on the ATS962x Waveform Digitizer, namely the Trigger input, the Auxiliary 1 or Auxiliary 2. Connecting it into the Trigger input allows the same signal to be used for both trigger function and 1pps function without requiring an external cable splitter.

Note that there is no requirement for the GPS frequency to be 1 second, it can be any time base, as long as it is not too long as to overflow the 32-bit clock counters.

12.7.2 One-second Frequency Counter

This function is independent of the buffer delineation and timestamp, but it does require a GPS_in signal.

Upon every GPS_in rising edge, the Time Counter's value is latched in the freq_cnt register as it is reset. By reading this register, it allows the user to monitor the frequency deviation of the ADC clock versus the GPS clock.

For example, if the onboard clock oscillator was used and the freq_cnt register had a value of 250,000,250, it would reveal that the onboard clock oscillator was operating at 1ppm above its nominal 250MHz, when compared to the reference 1pps GPS_in signal.

© 2003-2012 Alazar Technologies Inc.

13 Memory Map

The Coprocessor FPGA is accessible by the host PC through the Main FPGA. The address space consists of 14 address bits, each addressing a 32-bit register. This provides a total of 16,384 individual 32-bit locations.

The memory map is divided into 3 global sections:

Table 13-1 Memory Map - Global

Addr (dec)	Section Name	Description
0 - 31	**Alazar section (Reserved)**	This is a fixed section which the user must not modify.
32-127	**User section, Alazar-Defined**	This section is driven by circuitry in the user section, but the definition and locations of the registers is defined by Alazar.
128-16383	**User section, Open**	This is open for the user to modify freely.

© 2003-2012 Alazar Technologies Inc.

13.1 Memory Map - Alazar Section (Reserved)

The Alazar section of the memory is a reserved section of 32 locations that the user must not change. Bit definitions or special functions are further detailed in subsections below, per register.

Table 13-2 Memory Map - Alazar Reserved Section

Addr (dec)	Register Name	RW	Description
0	cpf_signature	R	An Alazar-internal unique identifier for the Coprocessor FPGA. Do not change. (1234_5678h)
1	cpf_version	R	The major and minor version of the CPF Alazar framework code release. Do not change.
2	cpf_variant	R	Upper 2 bytes define the variant of the cpf framework used, namely Standard or Advanced. Year timestamp is also included in the lower 2 bytes.
3	compilation_timestamp	R	Date & Time timestamp which gets auto-updated everytime the design is compiled in Quartus. Provides a truly unique identifier (MM:DD:HH:mm)
4	scratchpad	RW	A 32-bit RW test register which has no connection to circuitry other than the scratchpad_inv register.
5	cpf_config	RW	The main CPF Configuration register.
6	cpf_status	R	The main CPF Status register.
7-11	-- Reserved for future use --		
12	scratchpad_inv	R	A read-only register which shows the inverted value of the contents of the scratchpad register.
13-31	-- Reserved for future use --	-	

13.1.1 cpf_version [Addr 1]

Bit	Bit Name	RW	Rst	Description
31-16	<reserved>	R		Reserved.
15-8	cpf_ver_major	R		The 8-bit value of the major version.
7-0	cpf_ver_minor	R		The 8-bit value of the minor version.

The major and minor version of the CPF Alazar framework code release. Do not change. Please use the version register in the User section for your versioning purposes.

13.1.2 cpf_variant [Addr 2]

Bit	Bit Name	RW	Rst	Description
31-16	cpf_variant	R		The 16-bit value of the cpf framework variant. 0000: Standard framework (cpf) 8000: Advanced framework (cpf_adv)
15-0	year	R		The 16-bit value of the build timestamp year version, in a hex-digit to dec-digit format.

For the "year" description and format, refer to the compilation_timestamp Register Description.

Example: 80002012h: Variant 8000 (cpf_adv), Build timestamp year 2012.

© 2003-2012 Alazar Technologies Inc.

13.1.3 compilation_timestamp [Addr 3]

This register is automatically updated when compiling the design in Quartus, and provides a unique identifier for every build that is made. This is a convenient feature to differentiate between multiple debug compiles of the same version, moreover it provides the exact time at which the load was compiled so makes it easy to trace back to source code changes.
The format is as follows: it should be read out of the register and displayed in a 32-bit hex format. Then each byte holds a value that should be looked at as a decimal number. Thus by reading the register in its 32-bit hex value, a human reader can very quickly interpret the compile time. For clarification:

09151005h: September 15th, 10:05 (10:05am)
10281759h: October 28th, 17:59 (5:59pm)
01020001h: January 2nd, 0:01 (12:01am (1 minute past midnight))

Bit	Bit Name	RW	Rst	Description
31-24	month	R		Month (01-12).
23-16	day	R		Day of month (01-31)
15-8	hour	R		Hour (24-hour format) (00-23)
7-0	minute	R		Minute (00-59)

13.1.4 cpf_config [Addr 5]

Bit	Bit Name	RW	Rst	Description
31-1	<reserved>			
0	user_reset	RW	0	Active High signal which gets synchronized to the i_user_adc_clk domain and goes to user_top_level. Should be used to reset user circuitry. Held active during board reset, auto-cleared a few cycles after.

13.1.5 cpf_status [Addr 6]

Bit	Bit Name	RW	Rst	Description
31-2	<reserved>	R	0	
1	pll50_locked	R	0	Active High signal which indicates that the PLL fed by the external 50MHz clock oscillator is locked. Provided by user logic; disregard if you choose to not use this PLL.
0	user_circuit_is_alive	R	0	Active High signal asserted by the user logic to indicate that it is out of reset and running, PLL's locked, and any other prerequisites the user may deem important.

© 2003-2012 Alazar Technologies Inc.

13.2 Memory Map - User Section, Alazar-Defined

The user section, Alazar-Defined registers range from address 32 to 127 decimal. The circuitry for these registers is in the user section, but they are defined by Alazar so software can interpret the correct information depending on the nature of the CPF functionality. The address map is defined in VHDL in the file called "user_mem_map_pkg.vhd".

Table 13-3 Memory Map - User Customizable Section

Addr (dec)	Register Name	RW	Description
32	user_signature	R	A user-defined signature for the FPGA load, which can be used to identify different CPF designs and functionality. Update as needed. Default: 0000_0001h
33	user_version	R	The major and minor version of the User's design block. Update contents as needed.
34	scratchpad	RW	A 32-bit RW test register which has no connection to circuitry other than the scratchpad_inv register.
35	scratchpad_inv	R	A read-only register which shows the inverted value of the contents of the scratchpad register.
36	data_format_cha	R	A read-only register which shows Channel A's data format and width. This describes the data output from the cpf to the main FPGA.
37	data_format_chb	R	A read-only register which shows Channel B's data format and width. This describes the data output from the cpf to the main FPGA.
38-42	<not used in this design>		
43	bytes_per_buffer_out	RW	A software configurable 32-bit register which instructs the CPF FPGA how many bytes of data are output per buffer. Its value does NOT include the header. This value must be an even multiple of 32, for system reasons.
44-46	-- Reserved for future definition by Alazar --	-	
47	<not used in this design>	R	
48-127	-- Reserved for future definition by Alazar --	-	

13.2.1 user_version [Addr 33]

The VHDL definition of this register is located in user_version_registers_pkg.vhd. Update as needed.

Bit	Bit Name	RW	Rst	Description
31-16	user_ver_major	R		The 16-bit value of the major version.
15-0	user_ver_minor	R		The 16-bit value of the minor version.

© 2003-2012 Alazar Technologies Inc.

13.2.2 data_format_cha/chb [Addr 36, 37]

This register is used by Alazar software to properly configure the Main FPGA's acquisition parameters. It is important that the values of these registers properly reflect the actual data formats used for the acquisition data to be properly transferred.

Bit	Bit Name	RW	Rst	Description
31	Endianness	R	0	0: Little Endian, 1: Big Endian Note: Currently only Little Endian is supported.
30-16	data_format_type	R	00	00h: Unsigned Integer 01h: Signed Integer 02h: Floating Point (per IEEE 754) 03h: Custom
15-0	data_format_width	R	16d	The number of bits on the c2m_data bus used up by each sample. 00h: Channel disabled 10h: 16 bits 20h: 32 bits

Currently Supported Data Formats:

0000_0010h 16-bit Unsigned Integer.
0000_0020h 32-bit Unsigned Integer.
0001_0010h 16-bit Signed Integer.
0001_0020h 32-bit Signed Integer
0002_0020h 32-bit Floating-Point (single-precision IEEE754)

13.2.3 bytes_per_buffer_out

This register is used when Header insertion is enabled. It tells the FPGA how many bytes of data are contained in a buffer. The buffer size specified in this register does not include the buffer's header.

This value must be an even multiple of 32, for internal system reasons.

Note that each sample of each channel is 16-bit (2 bytes). Therefore, if one wants a buffer containing 1024 samples from each of 2 channels, one would calculate:

1024 samples x 2 bytes/sample x 2 channels = 4096 bytes per buffer.
One would write 4096dec (1000 hex) into the bytes_per_buffer_out register.

The 32-byte header, when enabled, would be pre-pended to each of these 4096 bytes, for a total of 4096+32 = 4128 bytes for software to analyze as a data structure.

Please also refer to section 12.6, Buffer Header (optional function).

© 2003-2012 Alazar Technologies Inc.

13.3 Memory Map - User Section, Open (Customizable)

The user open section starts at address decimal 128 and goes up to the top of the available addresses (see table). This address map is defined in VHDL in the file called "user_mem_map_pkg.vhd" and can be customized by the user as desired.

Table 13-4 Memory Map - User Customizable Section

Addr (dec)	Register Name	RW	Description
128	user_cfg	RW	Configuration register, which contains bits to configure the user circuit, for example datapath multiplexer selectors.
129	user_status	R	Suggested status register, which contains bits to monitor the status of the user circuit.
130	latch_regs	RW	Contains 1 active bit which updates internal counters into readable registers indicated by (latched reg).
131	nco_phase_incr	RW	NCO configuration register: Phase increment, sets the NCO's frequency which is used for demodulation.
132	user_acq_cfg	RW	Acquisition Configuration register.
133	acq_mgr_byte_cntr (latched register)	R	Latched value of the byte counter in the acquisition manager (for debug purposes). Latched using latch_regs.
134	acq_mgr_word_cntr (latched register)	R	Latched value of the 16-bit word counter in the acquisition manager (for debug purposes). Latched using latch_regs.
135	buf_seq_num_cntr (latched register)	R	Latched value of the Buffer sequence number counter in the acquisition manager (for debug purposes). Latched using latch_regs.
136-139	-- unassigned --	-	
140	fir_coeff_cfg	RW	FIR coefficient configuration interface: control and status bits to read and write the coefficients in the FIR filters.
141	fir_coeff_addr	RW	FIR coefficient configuration interface: address of the coefficient for subsequent read or write operation.
142	fir_coeff_in_data	RW	16-bit coefficient data to be written
143-146	fir_coeff_out_data(4..1)	R	Coefficient data read from the 4 FIR filters, in respective order.
147	rescaler_slice_pos	RW	The FIRs' output is much wider than 16 bits. Use this register to specify the slice's msb position in the larger output word. The smallest allowed value is 15, which indicates to take the least significant bits 15 downto 0.
148	gps_freq_cnt (latched register)	R	This is a latched copy of the GPS frequency counter that is reset every GPS input edge. It holds the last value of the GPS time counter (which increments every ADC clock cycle) before it got reset by the GPS edge. If the GPS outputs 1 pulse per second, then this value can be used to measure the ADC clock's frequency relative to this 1 second reference. Use latch_regs to update the readable register.

© 2003-2012 Alazar Technologies Inc.

Addr (dec)	Register Name	RW	Description
149-16383	-- For user's purposes --	-	For the user to define and use as needed.

13.3.1 user_cfg [Addr 128]

This register is meant as a general-purpose configuration register, at the user_top_level hierarchy level. Please also refer to the FPGA Block Diagram for a graphical view.

Bit	Bit Name	RW	Rst	Description
31-20	-- unassigned --	-	0000	
19-16	cfg_gps_in_sel	RW	0	Select GPS input port on the faceplate. 0: TRIG 1: AUX1 2: AUX2
12	cfg_header_ena	RW	0	Enable header insertion at the beginning of every buffer. 0: No header 1: Header insertion enabled WARNING: Only enable when selecting a Filter path in cfg_data_out_sel, otherwise data corruption will occur.
4	cfg_demod_bypass	RW	0	Bypass the demodulation circuit, and route the ADC samples directly into the Filter block. 0: No bypass 1: Bypass

© 2003-2012 Alazar Technologies Inc.

Bit	Bit Name	RW	Rst	Description
3-0	cfg_data_out_sel	RW	0x0	Selects the datapath to output. **0:** Lower 2 Filter output channels (32-bit output). When cfg_demod_bypass=0, corresponds to the filtered, in-phase demodulation of channels A and B. When cfg_demod_bypass=1, corresponds to the filtered channels A and B. **1:** Upper 2 Filter output channels (32-bit output). Corresponds to the filtered, quadrature-phase demodulation of channels A and B. **2:** All 4 Filter output channels (64-bit output). Lower 2 channels output first. **4:** Lower filter output channel (16-bit output). When cfg_demod_bypass=0, corresponds to the filtered, in-phase demodulation of channel A. When cfg_demod_bypass=1, corresponds to the filtered channel A. **5:** 2nd filter output channel (16-bit output). When cfg_demod_bypass=0, corresponds to the filtered, in-phase demodulation of channel B. When cfg_demod_bypass=1, corresponds to the filtered channel B. **6:** In-phase demodulation (I) of channels A and B, unfiltered. (32-bit output) **7:** Quadrature-phase demodulation (Q) of channels A and B, unfiltered. (32-bit output) **D:** NCO's raw cos and sin outputs. (32-bit output) **F:** Transparent output (32-bit output). Channel A & B data from the ADCs.

The table below provides a summary of settings and the resulting output data format:

© 2003-2012 Alazar Technologies Inc.

cfg_data_out_sel	cfg_demod_bypass	Selected Output	Chan	Sample Rate	32-bit output order [MSb...LSb]
0	0	DDC I, filtered	A&B	1/3	[ChB_I, ChA_I]
0	1	FIR mode: ChA & ChB, filtered (no DDC)	A&B	1/3	[ChB_I, ChA_I]
1	0	DDC Q, filtered	A&B	1/3	[ChB_Q, ChA_Q]
2	0	DDC I&Q, filtered	A&B	1/3	[ChB_I, ChA_I] [ChB_Q, ChA_Q]
4	0	DDC I, filtered	A	1/3	[ChA_1, ChA_0]
4	1	FIR mode: ChA, filtered	A	1/3	[ChA_1, ChA_0]
5	0	DDC I, filtered	B	1/3	[ChB_1, ChB_0]
5	1	FIR mode: ChB, filtered	B	1/3	[ChB_1, ChB_0]
6	0	DDC I, unfiltered	A&B	1	[ChB_I, ChA_I]
7	0	DDC Q, unfiltered	A&B	1	[ChB_Q, ChA_Q]
D	0	NCO's raw cos and sin output		1	[NCO_{SIN}, NCO_{COS}]
F	0	Transparent	A&B	1	[ChB, ChA]
others	others	not supported			

Table 13-5 Summary of user_cfg Settings and Output Data Format

13.3.2 user_status [Addr 129]

This register is meant as a general-purpose status register and currently has no bits assigned.

Bit	Bit Name	RW	Rst	Description
31-0	-- unassigned --	-		

13.3.3 latch_regs [Addr 130]

Use this register to latch internal counters' values into readable registers which are indicated as "latched register" in the memory map table.

This register contains a single bit. When it is high, data for all registers indicated by "latched_register" is copied every clock cycle from the associated internal counter to the readable register. When it goes low, the last value is retained (latched) in a guaranteed-coherent manner.

Bit	Bit Name	RW	Rst	Description
31	latch_regs	RW	0	Latch internal counters into readable, latched registers. 0: latched registers are ready to read. 1: update latched registers with new values
30-0	-- unassigned --	-		

© 2003-2012 Alazar Technologies Inc.

13.3.3.1 About latched registers

Since the internal counters change rapidly and asynchronously to a host read operation, there is a high likelihood that the counter values will change during the read operation, resulting in a mix of old and new bits which will look like data corruption.

The correct way to read "latched_register" registers is to do the following:

1. Set latch_regs to 1.
2. Set latch_regs to 0.
3. Read any/all latched registers of interest. Their values are guaranteed coherent and will not change until the next time the latch_regs bit is set high.

13.3.4 nco_phase_incr [Addr 131]

This register configures the NCO's phase increment value, which directly translates into its output frequency, and thus the demodulation frequency.

Bit	Bit Name	RW	Rst	Description
31-0	nco_phase_incr	RW	1,202,590,843	See below

The phase increment value is a 32-bit number, and its value can be determined by the following formula, for a desired frequency f_o, with the ADCs running at f_{clk}:

$$phase\_incr = 2^{32} \times f_o / f_{clk}$$

The nominal ADC clock frequency is 250MHz, but is dependent on user settings; please refer to section 11.3.2 for more information.
Example rates and values, based on 250MHz:

70MHz: 1,202,590,843 decimal, 47AE147B hex
20MHz: 343,597,384 decimal, 147A E148 hex

13.3.5 user_acq_cfg [Addr 132]

This register is the acquisition configuration register, and mainly controls the Acquisition Manager. Please refer to the Acquisition Manager functional description for operation details. Please also refer to the FPGA Block Diagram for a graphical view.

Bit	Bit Name	RW	Rst	Description
31-30	-- unassigned --	-		
29	force_trig	RW	0	Forces a Trigger event upon transitioning from 0 to 1 (regardless of trigger polarity). Use to force a trigger in the acquisition manager via software.
28-21	-- unassigned --	-	0	
20	trig_negedge	RW	0	Determines the edge polarity detected by the Acquisition Manager for the Trigger input on the Front Panel connector. 0: Trigger positive (rising) edge 1: Trigger negative (falling) edge
19	-- unassigned --	-	0	

© 2003-2012 Alazar Technologies Inc.

Bit	Bit Name	RW	Rst	Description
18	triggered_controlled	RW	0	Turns on Trigger-Controlled operation. 0: Free-running acquisition 1: Trigger-Controlled acquisition
17-5	-- unassigned --	-	0	
4	gps_negedge	RW	0	Determines the edge polarity detected by the GPS Timestamp engine. 0: GPS positive (rising) edge 1: GPS negative (falling) edge
3-0	-- unassigned --	-	0	

13.3.6 acq_mgr_byte_cntr [Addr 133] (Latched Register)

This is a latched register, and thus it is only updated when the latch_regs bit is toggled. Please refer to the latch_regs register for details.

This counter counts the number of bytes within a buffer.
For debug purposes.

13.3.7 acq_mgr_word_cntr [Addr 134] (Latched Register)

This is a latched register, and thus it is only updated when the latch_regs bit is toggled. Please refer to the latch_regs register for details.

This is the counter which counts the number of 16-bit words within a buffer.
For debug purposes.

13.3.8 buf_seq_num_cntr [Addr 135] (Latched Register)

This is a latched register, and thus it is only updated when the latch_regs bit is toggled. Please refer to the latch_regs register for details.

This counter counts the buffer sequence number which gets inserted in the header, when enabled.
For debug purposes.

13.3.9 fir_coeff_cfg [Addr 140]

This register allows reading and writing to the 4 FIR filters coefficients.

Bit	Bit Name	RW	Rst	Description
31-12	-- unassigned --	-		
11-8	fir_coeff_out_valid(4..1)	R	0	Read data valid. When set, coefficient data from the presented address from fir_coeff_addr is valid on the associated bits' filter(s).
7-4	fir_coeff_read(4..1)	RW	0	Read-enable. When set, coefficient data is read from each selected FIR filter's presented address from fir_coeff_addr. Any number of read bits can be set at the same time. Output data is presented on the corresponding fir_coeff_out_data registers.

© 2003-2012 Alazar Technologies Inc.

Bit	Bit Name	RW	Rst	Description
3-0	fir_coeff_we(4..1)	RW	0	Write-enable. When set, coefficient data from fir_coeff_in_data gets written into the associated filter(s) at the presented address from fir_coeff_addr. Any number of we bits can be set at the same time.

13.3.9.1 FIR Coefficient Read / Write Accesses

The filters are numbered 1 to 4 starting from the bottom, as indicated in the block diagram. All filters are identical in structure, and may have the same or different coefficients, as desired for the customer's application.

Each filter has its own write-enable (fir_coeff_we(4), fir_coeff_we(3), fir_coeff_we(2), fir_coeff_we(1)). A write is performed in the following sequence:

1. Write the desired address in fir_coeff_addr register
2. Write the desired data in fir_coeff_data_in register
3. Set the fir_coeff_we bit of all the filters which you wish to write to. Any/all can simultaneously be asserted.
4. Clear the fir_coeff_we bits.

In this manner, each coefficient can be written simultaneously to the 4 filters at once.

A read access is performed in the following sequence:

1. Write the desired address in fir_coeff_addr register
2. Set the desired fir_coeff_read bits (any/all can simultaneously be asserted).
3. Confirm that the desired fir_coeff_out_valid(s) is high
4. Read the desired data in the fir_coeff_out_data registers

13.3.10 fir_coeff_addr [Addr 141]

This register is used to indicate which coefficient will be accessed. The coefficients are addressed in order, starting at 0. Thus a 63-coefficient FIR's coefficients are addressed from address 0 to 62. Refer to fir_coeff_cfg [Addr 140] for more details.

Bit	Bit Name	RW	Rst	Description
31-12	-- unassigned --	-		
11-0	fir_coeff_addr	RW	0	Coefficient address, starting at zero.

13.3.11 fir_coeff_in_data [Addr 142]

This register is used to store the data which will be written to the FIR filter(s). Refer to fir_coeff_cfg [Addr 140] for details.

Bit	Bit Name	RW	Rst	Description
31-16	-- unassigned --	-		
15-0	fir_coeff_in_data	RW	0	Coefficient write data.

© 2003-2012 Alazar Technologies Inc.

13.3.12 fir_coeff_out_data(4..1) [Addr 143-146]

This register shows the read data supplied by the FIR filters. Refer to fir_coeff_cfg [Addr 140] for details.

Bit	Bit Name	RW	Rst	Description
31-16	-- unassigned --	-		
15-0	fir_coeff_out_data	R	0	Coefficient read data.

13.3.13 rescaler_slice_pos [Addr 147]

Each FIRs' output is scaled up by the many coefficient multiplications and is output in full-scale, which is much wider than the 16 bit datapath; in the default implementation it is actually 38 bits wide. The post-FIR rescaler scales this 38-bit value into a 16-bit value by taking a 16-bit slice in the 38-bit word, at a programmable slice position. This is equivalent to an integer division by a power-of-2 factor.

Use this register to specify the 16-bit slice's most-significant bit position in the larger output word. Bit positions are numbered starting at 0. Thus the smallest allowed value is 15, which indicates to take the least significant bits 15 downto 0. The largest is 37.

If coefficients are scaled appropriately (using a power-of-2 instead of full-range), it is possible to achieve near-unity gain from the Filtering block by using a corresponding rescaler value. This is the case with the default values.

13.3.14 gps_freq_cnt [Addr 148] (latched register)

This is a latched register, and thus it is only updated when the latch_regs bit is toggled. Please refer to the latch_regs register for details.

See description in the memory map table and in section 12.7.2 (One-second Frequency Counter).

© 2003-2012 Alazar Technologies Inc.

14 Special Considerations

14.1 Acquisition Mode

At a system software level, for the FIR/DDC FPGA to work properly, the ATS962x Waveform Digitizer's Acquisition Mode must be set to Coprocessor-Managed, via the ATS-SDK or AlazarDSO.

Please make sure the Acquisition Mode is set to Coprocessor-Managed!
With any other mode, many problems will occur, such as no data coming through, sample alignment issues within the records, etc.

The only case where an Acquisition Mode other than Coprocessor-Managed may be used is when selecting Transparent for the data output.

The Acquisition Modes and their uses are described in detail in the ATS962x Coprocessor FDK-Designer's Guide.

14.2 Data latency

The data incurs a processing delay as it goes through the filtering block. There is also latency in the Timestamp creation, and a small latency from the Acquisition Manager's trigger processing.

In order to keep the reference design simpler, these latencies are not internally compensated. Please review the consequences outlined below for your application.

14.2.1 Data latency vs Trigger

The trigger processing latency is quite short compared to the data latency. The acquisition manager will start outputting data a few cycles after receiving the trigger edge. Since the datapath latency is longer, this implies that the output data will be the filtered result from data that was acquired by the ADC prior to the trigger event occuring. In other words, there will be a constant pre-trigger value.

14.2.2 Data latency vs Timestamp

The Timestamp creation and the filtered data each have a processing latency.
Since these latencies are not internally compensated, this means the timestamp's absolute value will have an constant offset.

14.2.2.1 Consequence for Trigger-Controlled operation

One consequence of this data latency vs Timestamp in Trigger-Controlled operation, is that if the user uses the GPS 1pps signal for both Trigger and 1pps signal, the first buffer will not have a zero value, but will have this constant offset value.

© 2003-2012 Alazar Technologies Inc.

15 Quick Start Guide

AlazarDSO is a good platform for quickly configuring and observing the CPF FPGA processing results. AlazarDSO has both time-domain (Oscilloscope) and Frequency-Domain (Spectrum Analyzer) display capabilities.

Here is a summary of settings to quickly get started using the DDC/FIR FPGA. These same configurations can also be used with the ATS-SDK Alazar API's.

Configuration	**Setting**	**AlazarDSO Menu**	**CPF FPGA Addr=Data**
Download FIR/DDC CPF FPGA	(the default CPF FPGA does not have the FIR/DDC functionality!)	Tools-Coprocessor, Download (.rbf file)	
Clock Source/Sampling rate	250MHz	Configure-Horizontal	
Samples per record	16384 + (4 headers x 32)= 16,512	Configure-Acquire	
Acquisition Mode	Coprocessor Passthrough	Configure-Acquire AutoDMA Options Mode: Coprocessor Passthrough	
Enable both channels (required even if only using one)	Ch1: Not Disabled Ch2: Not Disabled	Configure-Input	
Spectrum Analyzer view		View-Spectrum Analyzer	
Spectrum Analyzer FFT parameters: 64k Blackman-Harris	64k, Blackman-Harris (choose as desired)	Configure-View	

Table 15-1 Quick Start Configuration Summary

Below is a table which summarizes typical FPGA settings.

Configuration	**Setting**	**DSO Menu**	**CPF FPGA Addr=Data**
FIR only (bypass DDC)	(as desired)	Tools-Coprocessor	`128 = 000x x010`
Header Enable	(as desired)	Tools-Coprocessor	`128 = 000x 10xx`
Triggered Operation	(as desired)	Tools-Coprocessor	`132 = 0004 0000`
Transparent Datapath (Direct ADC Data)	(as desired)	Tools-Coprocessor	`128 = 000x x0xF`

Table 15-2 FPGA Settings Summary

© 2003-2012 Alazar Technologies Inc.

15.1 Downloading the FIR/DDC FPGA

AlazarDSO's default CPF FPGA does not have the FIR/DDC functionality! You must download that FPGA load every time you reopen AlazarDSO or you reinitialize the board.

The FPGA load is a binary file which has the .rbf extension, and is an output from the Altera Quartus FPGA compiler. The pre-compiled file is provided in the FIR/DDC package under the fpga_binaries directory.
SL50 standard FPGA version:
ats9625_cpf_singleclk_v<n.m>_fir_ddc_v<x.y>.rbf
SE260 High-Capacity FPGA version:
ats9625_cpf**hc**_singleclk_v<n.m>_fir_ddc_v<x.y>.rbf

In AlazarDSO, it is downloaded in the following manner:

1. Press F5 (or select the Tools-Coprocessor menu) for Coprocessor Window
2. Press the Download button
3. Browse to the desired .rbf file and press Open.

15.2 Read/Write Accesses to CPF FPGA

Accessing the CPF FPGA registers is done as follows

1. Ensure the FIR/DDC CPF FPGA was downloaded as described in section 15.1.
2. Press F5 (or select the Tools-Coprocessor menu) for Coprocessor Window
3. In Address, type the decimal address128 (for example)
4. If Read is desired, press Read button.
5. If Write is desired, in Value type the hex 32-bit value: 00000010 (for example).
6. Press Write button.

To read or modify the FIR filter coefficients, please refer to Section 13.3.9.1.

15.3 Settings: DDC & FIR operation

By default, the FIR/DDC CPF FPGA starts up in DDC operation.

1. Ensure the FIR/DDC CPF FPGA was downloaded as described in section 15.1.
2. Perfom all AlazarDSO configurations from Table 15-1.
3. Press the Acquire ("Play") button in the main window.

For different output options, refer to the Memory Map section.
To read or modify the FIR filter coefficients, please refer to Section 13.3.9.1.

© 2003-2012 Alazar Technologies Inc.

15.4 Settings: FIR-only operation

For FIR-only operation (with x3 decimation), use the following settings:

1. Ensure the FIR/DDC CPF FPGA was downloaded as described in section 15.1.
2. Perform all configurations from Table 15-1;
   -> ensure you don't forget to set the mode to Coprocessor-Managed!
3. Press F5 for Coprocessor Window
4. In Address, type 128; in Data type 00000010.
5. Press Write
6. Press the Acquire ("Play") button in the main window.

15.5 Settings: Header Insertion Enable

Please note that header insertion adds data to the stream, and therefore is only supported on Filtered datapaths. Please refer to section 12.6.1 for details.

1. Use setting for either DDC&FIR, or FIR-only operation.
2. Press F5 for Coprocessor Window
3. In Address, type 128; press Read.
4. In Value, leave the non-X fields as is, and change the 1: xxxx1xxx.
   (Typically you'd have either 00001000 or 00001010).
5. Press Write
6. Press the Acquire ("Play") button in the main window.
7. Note that the header insertion adds header information to the datastream, thus causing discontinuities in the time and frequency domain. Data should be analyzed appropriately.
8. One can view the Hex values of data in AlazarDSO by doing:
   Configure-View-Oscilloscope tab: check the Display extended waveform Tooltips. The acquisition must be stopped for this to show.
9. Acquire and stop acquisition, zoom in to data and hover the mouse pointer over samples. Also, setting Configure-Input-Display Style to "Lines with Markers" helps hone in on specific samples.

Below is a screenshot of the header with waveform tooltips showing the hex value 0002 at sample number 1033 decimal.

© 2003-2012 Alazar Technologies Inc.

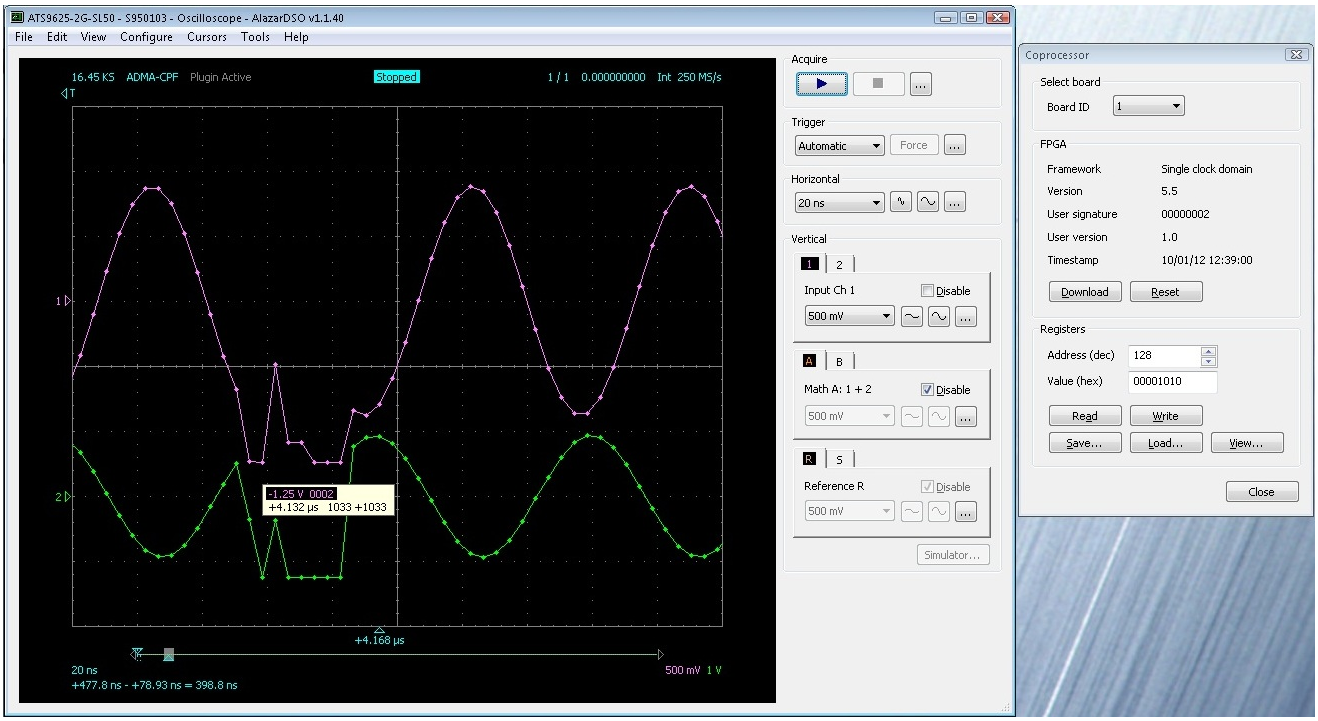


Figure 15-1 AlazarDSO Oscilloscope view, with Header Insertion

15.6 Settings: Transparent

For transparent operation, in which the data is fed through the CPF FPGA unmodified, do the following:

1. Perform all configurations from Table 15-1.
2. Write Addr 128, value 0000000F.
3. Press the Acquire ("Play") button in the main window.

Note: There is no trigger engine available in this setting. If you wish to get access to the standard AlazarDSO trigger engine, you must:

1. Set the AlazarDSO Acquisition Mode to: Enable dual-ported Memory (AutoDMA). Then click Options
2. Mode: No Pretrigger Samples, enable sample interleave
3. WARNING: Don't forget to set this back to Coprocessor Managed when going to another mode than Transparent!

15.7 Special Notes when using AlazarDSO with FIR/DDC FPGA

15.7.1 AlazarDSO Not Decimation-aware

AlazarDSO is not "aware" of the decimation factor and does not compensate for it. It always assumes the stream of samples is at the ADC clock rate. Therefore when data

obtained from a 3x decimated datapath is observed in AlazarDSO, one must mentally adjust the following:

- Time measurements are shown 3x shorter in DSO (1 ns shown in DSO is 3 ns)
- Frequency measurements are shown 3x faster in DSO (150kHz in DSO is 50kHz)

15.7.2 Channels 1&2 must always be enabled

In coprocessor-managed mode, Channels 1 & 2 must always be enabled in AlazarDSO. If channel 2 is disabled, channel 1 data will be corrupted. You may scale and move channel 2 out of the visible area however, as shown in the sample screenshot.

15.7.3 Sample screenshot

Below is a screenshot of the demodulation of an FM-modulated 50kHz signal on a 70MHz carrier. Notice the 150kHz measured which really represents 50kHz, since this is a stream of samples acquired at ADC Clock of 250MHz (250MS/s), decimated by 3 by the FIR filter (becoming 83.3MS/s). AlazarDSO shows it as if it was a 250MS/s stream.)

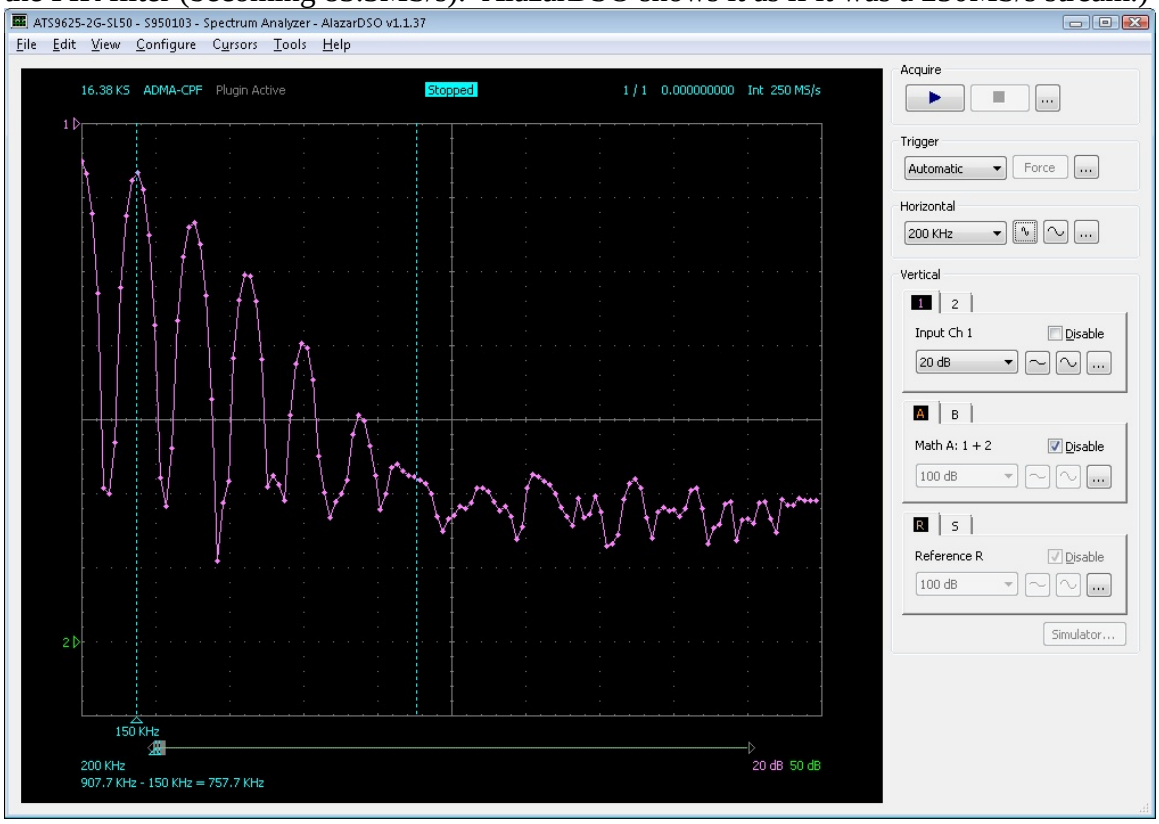


Figure 15-2 AlazarDSO Spectrum Analyzer view - Demodulated & Filtered data

One can obtain the above by configuring AlazarDSO per section 15.3, and connecting an FM-modulated 50kHz signal on a 70MHz carrier to Channel A. Zoom the horizontal and vertical scales as appropriate.

© 2003-2012 Alazar Technologies Inc.

Book III - MultiClk Framework and FFT Example

-- BOOK III --

FDK MultiClk Framework with FFT Example

© 2003-2012 Alazar Technologies Inc.

Book III Contents

16 Book III - Introduction (MultiClk & FFT) 114
16.1 Scope of this Document 114
16.2 Overview of the MultiClk Framework and FFT Example 114
16.2.1 Source Code 114
16.2.2 Suitability 114
16.2.3 Clock Domains 114
16.2.4 Typical Applications 115
16.3 FFT Example Overview 115
16.4 How to Use this Document 116
16.5 Software Requirements 116
16.6 Pre-requisites 116
16.7 Disclaimer 117
16.8 Contacting us 117
17 Functional Description 118
17.1 ATS962x Waveform Digitizer Board Block Diagram 118
17.2 Front Panel Connectors 118
17.3 FPGA Top-Level Block Diagram (ats9625_cpf_multiclk) 120
17.4 Datapath Flow 120
17.5 Acquisition Modes 121
17.5.1 Triggered Operation (Coprocessor-Managed Acquisition) 121
17.5.2 Continuous Operation (*non* Coprocessor-Managed Acquisition) 123
17.5.3 Transparent Operation 125
17.6 Auxiliary Ports 126
17.6.1 AUX1 (TrigOut) 126
17.6.2 AUX2 (Trigger_Enable) 127
18 Detailed Functional Description 128
18.1 Datapath Clock Domains 128
18.2 coprocessor.vhd Block 128
18.3 user_top_level.vhd Block 128
18.4 Waveform Generator 129
18.4.1 Waveform Generator Trigger 130
18.5 Input Selector (wav_gen_sel Mux) 130
18.6 Input U16 to S16 Converter 130
18.7 pre_trig_dsp.vhd Block 130
18.7.1 Sample Skip 131
18.8 Acquisition Manager (with Trigger Engine) 131
18.8.1 Triggering Features 132
18.8.2 Trigger_Enable Feature 132
18.8.3 State Flow 133
18.8.4 Functional Timing Diagram - Simple Example 133
18.8.5 Functional Timing Diagram - Detailed Example 134
18.8.6 Configuration Parameters (Acquisition Manager) 135
18.8.7 Triggered Streaming 136
18.8.8 Continuous Streaming Mode 136

© 2003-2012 Alazar Technologies Inc.

18.8.9 Acquire <n> records then stop 136
18.9 post_trig_dsp.vhd Block 137
18.9.1 Block Diagram 137
18.9.2 Dataflow 138
18.9.3 Window Function 138
18.9.3.1 Post-Window Function Re-Scaler 139
18.9.3.2 Window Function RAM - Viewing and Changing 139
18.9.4 FFT (Fast-Fourier Transform) Core 140
18.9.4.1 Configuration Parameters (FFT) 140
18.9.4.2 FFT Internal Scaling 140
18.9.5 Post-FFT Output Scaling & Calculations 141
18.9.6 post_trig_out Output Selector 141
18.10 Output Selector (data_out_sel Mux) 142
18.11 Output Smart S16 to U16 Converter 142
18.12 Reset Hierarchy 142
18.12.1 RAM contents not restored by Reset 143
18.13 Trigger Pulse Width Requirements 143
19 Memory Map 144
19.1 Memory Map - Alazar Section (Reserved) 145
19.1.1 cpf_version [Addr 1] 145
19.1.2 cpf_variant [Addr 2] 145
19.1.3 compilation_timestamp [Addr 3] 146
19.1.4 cpf_config [Addr 5] 146
19.1.5 cpf_status [Addr 6] 146
19.2 Memory Map - User Section, Alazar-Defined 147
19.2.1 user_version [Addr 33] 148
19.2.2 data_format_cha/chb [Addr 36, 37] 148
19.3 Memory Map - User Section, Open (Customizable) 149
19.3.1 user_cfg [Addr 128] 150
19.3.2 user_status [Addr 129] 151
19.3.3 datapath_reset [Addr 130] 151
19.3.4 wav_gen_cfg [Addr 131] 151
19.3.5 user_acq_cfg [Addr 132] 151
19.3.6 ram_ctrl [Addr 144] 153
19.3.7 window_fn_ram_addr/dout/din [Addr 145-147] 153
19.3.8 post_trig_cfg [Addr 160] 153
20 Miscellaneous Considerations 155
20.1.1 Datapath/Trigger/Aux Latency Matching Considerations 155
20.1.2 FPGA Timing Violations 155
21 Software and System Considerations 156
21.1 Acquisition Transfer Modes 156
22 FPGA Designer Section 157
22.1 Please read the FDK Designer's Guide 157
22.2 Functional Simulation 157
22.2.1 Complete FPGA Functional Simulation 157
22.2.2 FFT Core Functional Simulation 157

© 2003-2012 Alazar Technologies Inc.

22.2.3 Special Note on Functional Simulation Analog Waveforms 158
22.3 FPGA Code Structure 158
22.4 Data_Valid and Rate Adaptation 158
22.4.1 data_valid in post_trig_dsp 159
22.5 Clocks and PLLs 159
22.5.1 user_dsp_clk Frequency 159
22.5.2 Changing a PLL frequency 160
22.6 C2M Bus 160
23 Design Recommendations 161
24 Simulation Framework 162
25 Glossary 163

© 2003-2012 Alazar Technologies Inc.

List of Figures

Figure 17-1 ATS9625 Board Block Diagram 118
Figure 17-2 FPGA Block Diagram (ats9625_cpf_multiclk) 120
Figure 17-3 Triggered Datapath (Typical use) 123
Figure 17-4 Continuous Datapath (Typical use) 125
Figure 17-5 Transparent Datapath (Default at Startup) 126
Figure 18-1 Acquisition Manager Functional Timing Diagram (Simple) 134
Figure 18-2 Acquisition Manager Functional Timing Diagram (Detailed) 134
Figure 18-3 post_trig_dsp.vhd Block Diagram 137

List of Tables

Table 17-1 Front Panel Connectors 119
Table 19-1 Memory Map - Global 144
Table 19-2 Memory Map - Alazar Reserved Section 145
Table 19-3 Memory Map - User Customizable Section 147
Table 19-4 Memory Map - User Customizable Section 149

© 2003-2012 Alazar Technologies Inc.

16 Book III - Introduction (MultiClk & FFT)

This document describes the ats9625_cpf_multiclk (multiple clock domain) framework and example project, which is an advanced application project of the ATS962x Coprocessor FPGA Development Kit (FDK). Although much of the text refers to ATS9625, it also applies to all members of the ATS962x family, such as the ATS9626.

16.1 Scope of this Document

This document is meant as an addendum to the ATS962x Coprocessor FDK-Designer's Guide. The reader is expected to read the Designer's Guide document first, and this document next. This document further expands the bases and highlights differences in order to describe the ats9625_cpf_multiclk example project.

16.2 Overview of the MultiClk Framework and FFT Example

16.2.1 Source Code

It is worth noting that the MultiClk framework and the FFT example are provided as one single unit of source code. There is no "empty" MultiClk framework. The reason for this is that a certain set of architectural structures and functions from the FFT example may be desirable and reusable for some applications as part of the framework, while a different set may be useful for other applications.

Should the user want to base a design on the MultiClk framework, the desired blocks from the FFT example should be kept while the non-desired ones removed, based upon the user's needs.

16.2.2 Suitability

The multiple clock domain framework is more complex than the standard single-clock framework of the ATS962x Coprocessor FDK. As such it is recommended to be used only when the target application requires it.

In order to determine whether you need to use the multiple clock domains framework for you application, please refer to the ATS962x Coprocessor FDK-Designer's Guide, Design Recommendations Chapter, Clocking Schemes section.

16.2.3 Clock Domains

The multiple clock domain framework basically provides independent clock domains in the Coprocessor FPGA (CPF) for:

1. adc_clk: ADC input data to the CPF
2. user_dsp_clk: User's Digital Signal Processing core circuit
3. c2m_data_clk: CPF output data to the system (Main FPGA)

© 2003-2012 Alazar Technologies Inc.

Each clock domain is isolated through a FIFO buffer that stores data and trigger information. The user_dsp_clk and c2m_data_clk domains are flow-controlled and thus support different and/or variable throughput rates.

16.2.4 Typical Applications

The 3 independent domains provide the user with maximum flexibility. One can operate the ADC encode frequency completely independently of the actual digital signal processing frequency, and independently again from the output transfer frequency.

This allows:

- Variable-frequency acquisition from an externally-supplied clock.
- Maintaining maximum ADC encoding frequency (e.g. 250MHz), while using slower-frequency clock signal processing circuitry (e.g. 200MHz), thus allowing the use of more complex FPGA processing circuitry that cannot run at the full ADC clock frequency, and alleviating FPGA timing closure.
- Using a slower or varying ADC encode frequency (e.g. 100MHz) with faster signal processing frequency (e.g. 225MHz), allowing for easier implementations of interpolation for example, or simply providing more raw signal processing throughput.
- Maintaining maximum output data rate independently of ADC and processing frequencies. This allows a fast transfer rate and either data expansion (e.g. floating-point output, interpolation etc), or insertion of extra information in addition to the output samples.

16.3 FFT Example Overview

In addition to being a complete multiple clock domain FPGA development framework, the ats9625_cpf_multiclk example project is also a fully featured FPGA data processor, useable out-of-the-box to demonstrate practical real-world applications.

Summary of functionality:

- FFT of configurable length of up to 4096 points.
- Acquisition Management engine, highly configurable with support for Trigger, Trigger Enable inputs and providing Trigger Output. It supports configurable samples-per-record and records-per-buffer, various acquisition modes, software-triggering and trigger-inhibit, etc.
- Trigger has zero re-arm time, can acquire back-to-back records with no inter-record delay.
- Multiple selectable data outputs, including post-FFT's real, imaginary, amplitude, log of amplitude, with floating-point scaling at various levels. Pre-FFT data also available, such as pre-window and post-window.
- Window function at FFT input, fully-configurable through either software, "live" in-circuit programming, or at FPGA compile-time.
- RAM-based input Waveform generator: provides the FPGA with deterministic input data instead of the ADC's data, and is user-programmable. This is

© 2003-2012 Alazar Technologies Inc.

particularly useful for characterization and verification, software debugging, and is also useful for development without needing real hardware to generate input stimuli.
- Preselects half the FFT output record to cut out the redundant mirror-image FFT data at the source, saving transfer bandwidth on the PCI Express bus and subsequent software buffer transfer and processing.
- Transparent mode, whereby the CPF passes unmodified input data and trigger through to the Main FPGA.
- Software-accessible control and status registers
- Modular and well-structured FPGA source code, allowing the user to easily add or customize functionality.

16.4 How to Use this Document

If you intend on using the example MultiClk design as-is without modification, you can omit the FPGA Designer Section. The first sections of this document are structured in such a way that it can be used much like a datasheet.

If you intend on modifying the example design to add in your own custom functions, then it is highly recommended that you read the entire document AND, very importantly, the ATS962x Coprocessor FDK-Designer's Guide as well, before starting your custom work. Knowledge and understanding of the high-level and low-level design considerations will help make the right decisions from the start.

16.5 Software Requirements

The multiclk version of the ATS962x Coprocessor FDK requires the following software versions:

- ATS9625 (or ATS9626) driver v5.9.2 or higher (required for all applications).
- AlazarDSO v1.1.37 or higher (if using AlazarDSO).
- ATS-SDK v6.0.4 (if using an SDK).

The driver is required in all cases. The user has the choice of using an ATS-SDK or AlazarDSO as an interface, or both.

16.6 Pre-requisites

The pre-requisites for developing using the FPGA Development Kit framework are the same as specified in the ATS962x Coprocessor FDK-Designer's Guide. Please refer to the Pre-requisites section of that document.

Please note that the ats9625_cpf_multiclk framework, due to its multiple clock domains and more complex sample processing application, is inherently more complex as a whole than the single-clock ats9625_cpf framework.

© 2003-2012 Alazar Technologies Inc.

That being said, significant effort has been put in by Alazar Technologies to isolate and "abstract out" this additional complexity from the user, for most applications. Thus in most cases, the user's custom circuitry can be inserted into the design in a fairly straightforward manner, similar to the single-clock framework.

16.7 Disclaimer

AlazarTech, Inc. provides this FPGA design AS IS, WITHOUT ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. AlazarTech makes no guarantee or representations regarding the use of, or the results of the use of, the software and documentation in terms of correctness, accuracy, reliability, currentness, or otherwise; and you rely on the FPGA design, documentation and results solely at your own risk.

In no event shall AlazarTech be liable for any loss of use, loss of business, loss of profits, indirect, incidental, special or consequential damages of any kind. In no event shall AlazarTech's total liability exceed the sum paid to AlazarTech for the product.

16.8 Contacting us

Please contact us if you have any questions or comments about this document, or the sample code.

Web	http://www.alazartech.com
Email	mailto:support@alazartech.com
Phone	+1-514-426-4899
Fax	+1-514-426-2723
Mail	Alazar Technologies Inc. 6600 Trans-Canada Highway, Suite 310 Pointe-Claire, QC Canada H9R 4S2

Note that you can download the latest drivers and documentation here:
http://www.alazartech.com/support/downloads.htm

© 2003-2012 Alazar Technologies Inc.

17 Functional Description

The ats9625_cpf_multiclk example design is a fully featured FPGA data processor, useable out-of-the-box for some practical real-world applications. Please refer to section 16.3 for an overview of functionality.

This section provides technical details on the functionality and parameters for each of the functional blocks.

17.1 ATS962x Waveform Digitizer Board Block Diagram

Please refer to the ATS962x Coprocessor FDK-Designer's Guide for more details. The following figure is repeated here for convenience. The ATS9625 has AC-Coupled signal inputs, while the ATS9626 has DC-coupled inputs.

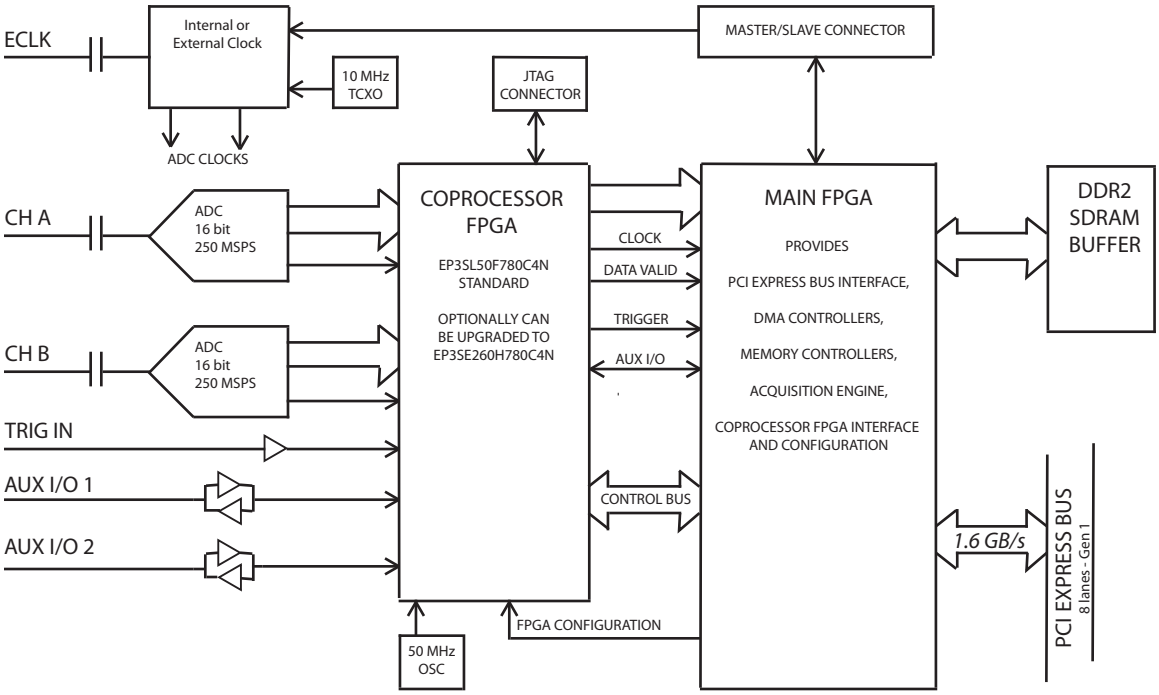


Figure 17-1 ATS9625 Board Block Diagram

17.2 Front Panel Connectors

The ATS962x Waveform Digitizer's front panel connectors are laser-etched to indicate their functionality. This table further clarifies their usage, specifically in the case of the MultiClk design, since the AUX1 and AUX2 are multifunction I/O's.

Table 17-1 Front Panel Connectors

Front Panel Name	Functional Name in MultiClk design	In/ Out	Description
ECLK	External Clock	In	External Clock input, which can optionally provide a source for the ADC's encode clock. Please refer to the ATS962x Waveform Digitizer documentation for details.
CH A	Channel A	In	The Channel A analog input signal.
CH B	Channel B	In	The Channel B analog input signal.
TRIG IN	Trigger In	In	External Trigger input signal.
AUX1	Trigger Out	Out	Trigger output signal. Pulses to indicate that the trigger engine has begun an acquisition. In Continuous modes, it is driven by the Main FPGA's trigger engine. In Triggered modes, it is driven by the Coprocessor FPGA's Acquisition Manager.
AUX2	TriggerEnable	In	TriggerEnable input signal.

© 2003-2012 Alazar Technologies Inc.

17.3 FPGA Top-Level Block Diagram (ats9625_cpf_multiclk)

The following figure provides a detailed visual overview of the entire ats9625_cpf_multiclk FPGA design.

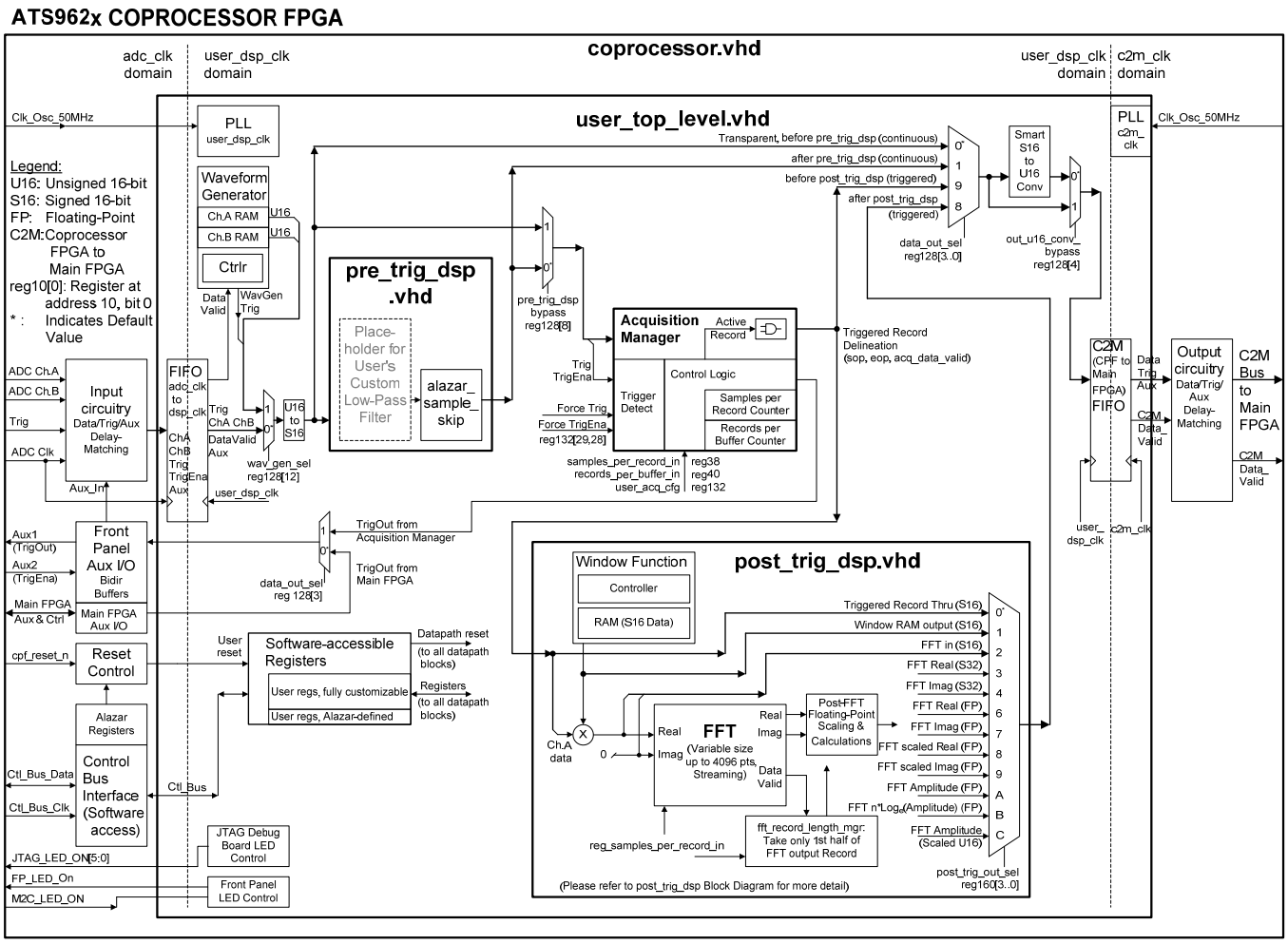


Figure 17-2 FPGA Block Diagram (ats9625_cpf_multiclk)

17.4 Datapath Flow

There a 4 major operating blocks on the datapath.

1) pre_trig_dsp: Pre-trigger processing block
2) Acquisition Manager: Record delineation. No processing.
3) post_trig_dsp: Post-trigger processing block.
4) Datapath selectors: Selects data routes in the FPGA.

The pre-trigger processing operates on the continuous incoming stream of data from the ADC's. It is the location where sample skipping is done (if enabled), and could be expanded to do such things as low-pass filtering etc.

© 2003-2012 Alazar Technologies Inc.

The Acquisition Manager delineates blocks of data, called "records", based on configuration parameters and the Trigger and Trigger_Enable input signals (from the faceplate connectors and/or software). It does not alter data in anyway, it simply provides delineation signals for the downstream datapath. For example, by default it delineates a record of 4096 samples every time a Trigger positive edge is detected.

The post_trigger processing operates on the samples in the records delineated by the Acquisition Manager. It is the location where record-based operations are performed. In the case of this design, a Fast-Fourier Transform (FFT) is performed on each record and a variety of output data processing is also performed and available for output. By default, a 4096-point FFT is performed on each 4096-sample record, and among others, the floating-point scaled Amplitude output is provided, with an output record size of 2048 points. The 2048-point output record size is half the input record size, since the FFT's output points 2049-4096 are a mirror image of the points 1-2048, and the design removes this redundant data.

The datapath selectors are the many multiplexers shown in the block diagrams. They select routes for the datapath, and can be configured by software for the desired operation.

Each block is described in more detail in their own section.

17.5 Acquisition Modes

The Coprocessor FPGA can be used mainly in 2 different ways:

- Coprocessor-Managed (for Triggered operation)
- non Coprocessor-Managed (for Continuous operation)

The Acquisition mode is selected in system software, using the appropriate method in AlazarDSO or in the ATS-SDK.

In the Coprocessor-Managed Acquisition mode, the Coprocessor FPGA takes charge of the trigger and data selection and processing.

In any Acquisition mode *other than* Coprocessor-Managed, it is the Main FPGA that is in charge of the trigger and data selection. The Coprocessor FPGA's role is to pre-process the ADC data as desired, but it must provide data to the Main FPGA in a continuous fashion.

The following subsections highlight the typical datapaths used in both cases.

17.5.1 Triggered Operation (Coprocessor-Managed Acquisition)

In triggered operation, the data does *not always* flow continuously through the coprocessor FPGA. The Acquisition Manager is enabled and determines precisely when

and how many data samples flow into the post_trig_dsp block, and ultimately through the Coprocessor FPGA to the software buffer.

The main concepts in this mode of operation is that the CPF is fully in charge of the acquisition and triggering. The CPF may transform, delete and create whatever data is desired. All CPF-validated output data bytes get transferred to the software buffer without further processing or selecting. The C2M bus effectively becomes a transparent 32-bit wide transfer bus which transfers data as-is from the CPF to the software buffer.

At a system software level, for CPF Triggered operation to work properly, the Acquisition Mode must be set to Coprocessor-Managed, via the ATS-SDK or AlazarDSO. With any other mode, many problems will occur, such as no data coming through, sample alignment issues within the records, etc.

Note: There are special End-of-Acquisition considerations in Coprocessor-Managed mode, to make sure all desired records are received.
It is strongly recommended that the user read the Software and System Considerations chapter in the ATS962x Coprocessor FDK-Designer's Guide, especially concerning the end of acquisition & buffer transfer considerations.

Typical applications using Triggered Operation include:

- Transforms, such as Fast-Fourier (FFT)
- Data Format conversion: for example converting the native U16 data to Floating-Point.
- Complete FPGA-based data processing, with only the final result getting transferred to software.
- Special Data insertion, such as inserting a computed average at the end of a record of samples
- Interpolation: high-ratio interpolation resulting in significant bandwidth increase for the span of each record. The inter-record gap provides extra bandwidth on the C2M bus for this extra data to get through before the next record.
- Custom trigger functionality
- Any combination of Continuous operation functions (listed in other section) and triggered operation functions.

Please refer to Figure 17-3 for the typical datapath used in Triggered operation.

© 2003-2012 Alazar Technologies Inc.

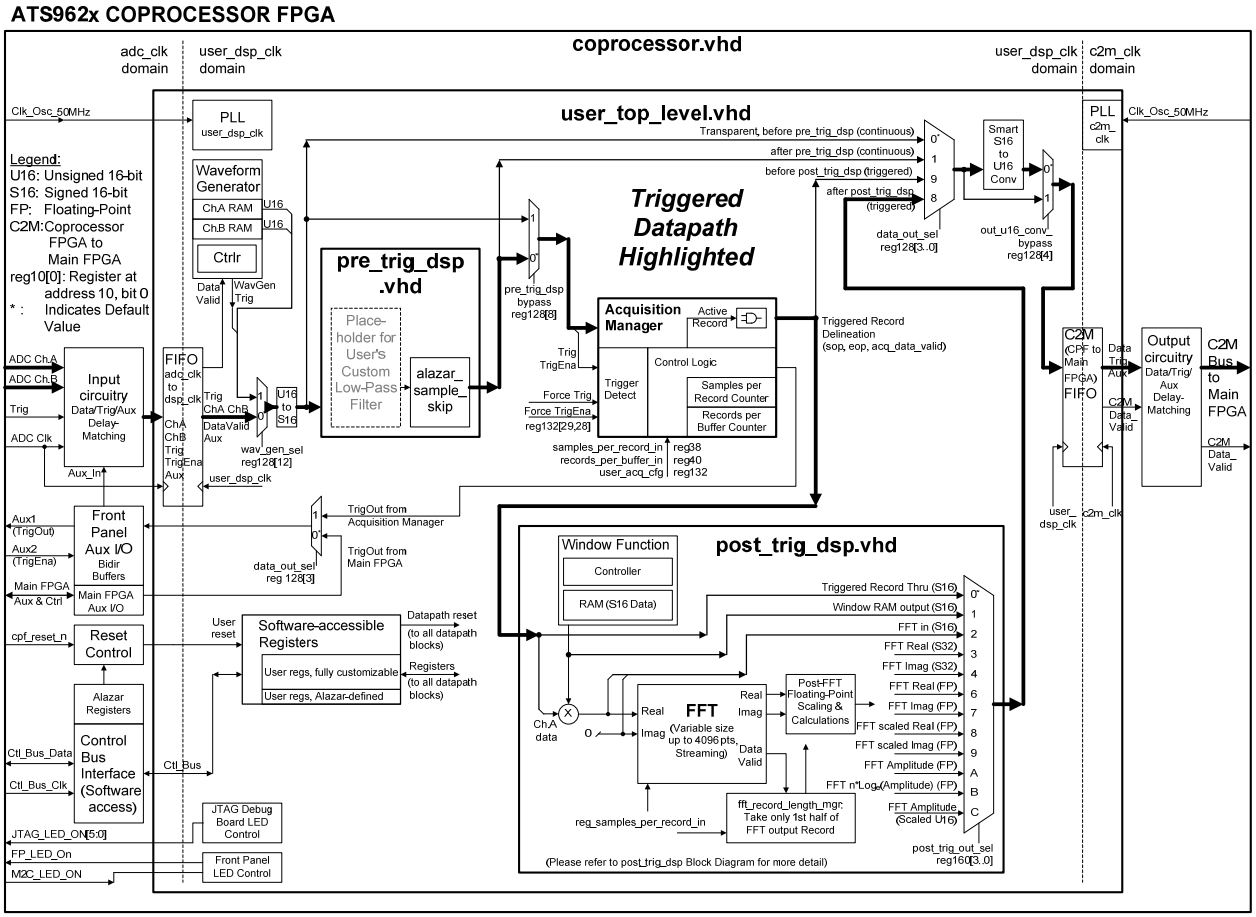


Figure 17-3 Triggered Datapath (Typical use)

The following register value is used to obtain this path:

- Register 128: 0x0000_0008

17.5.2 Continuous Operation (*non* Coprocessor-Managed Acquisition)

In continuous operation, the data flows continuously from the ADC's, through the CPF toward the Main FPGA. Data samples may be modified in various ways by the CPF, but there is no trigger handling nor record-based selection.

"Continuous data" can mean samples flowing at any sampling rate (including much reduced rates when decimating or down-converting for example), but the point is that it is not completely stopped at any point in time. By contrast, in Triggered Operation, there is no flow of data at all while the Acquisition Manager is waiting for a trigger. Data flow is therefore not continuous in Triggered Operation.

Please refer to Figure 17-4 for the typical datapath used in Continuous Operation.

In Continuous operation, only the pre_trig_dsp processing block may be used, if desired. The Acquisition Manager and post_trig_dsp are not used. For truly transparent operation, the path that taps before the pre_trig_dsp (on the data_out_sel mux) can be selected.

It is the Alazar Main FPGA's Trigger and acquisition engine that is used for managing the acquisition. It expects a continuous stream of data in the same format as the ADC provides, U16 (unsigned 16, offset at x"8000"), but it can be at any sampling rate.

At a system software level, for CPF Continuous operation to work properly, the Acquisition Mode must be set to any mode *other than* Coprocessor-Managed, via the ATS-SDK or AlazarDSO.

Typical applications include:

- Filtering
- Decimation
- Interpolation
- Digital Down-Converter
- Automatic Gain Control
- Transparent (no Coprocessor processing)

It is worth noting that all these functions can also be done in Triggered operation. If post-trigger (or record-based) processing is required, or custom triggering functionality, then Triggered operation must be used.

© 2003-2012 Alazar Technologies Inc.

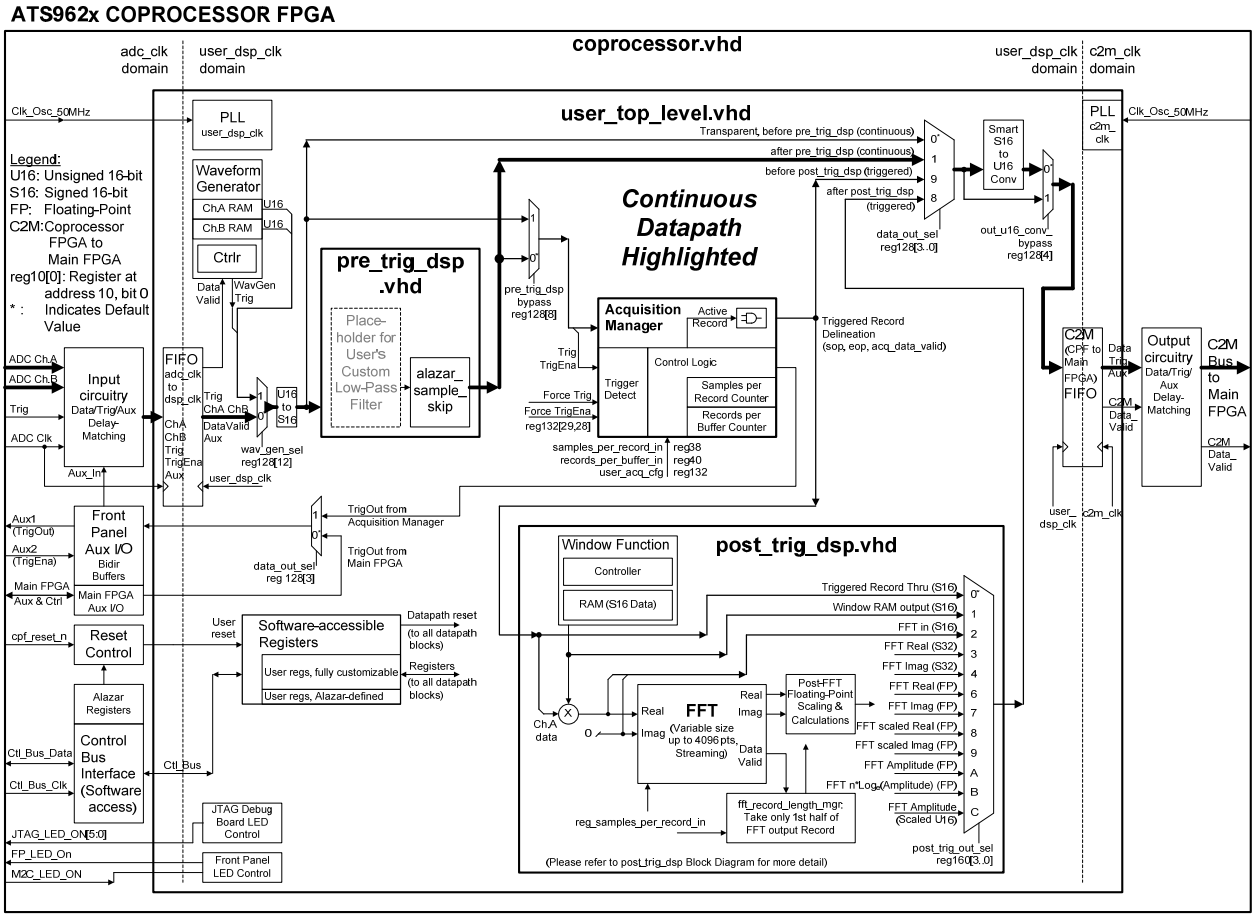


Figure 17-4 Continuous Datapath (Typical use)

The following register value is used to obtain this path:

- Register 128: 0x0000_0001

17.5.3 Transparent Operation

Transparent operation is a subset of continuous operation. It is a case where the Coprocessor FPGA effectively does nothing other than simply pass the signals through to the Main FPGA. Almost everything in the Coprocessor FPGA is bypassed.

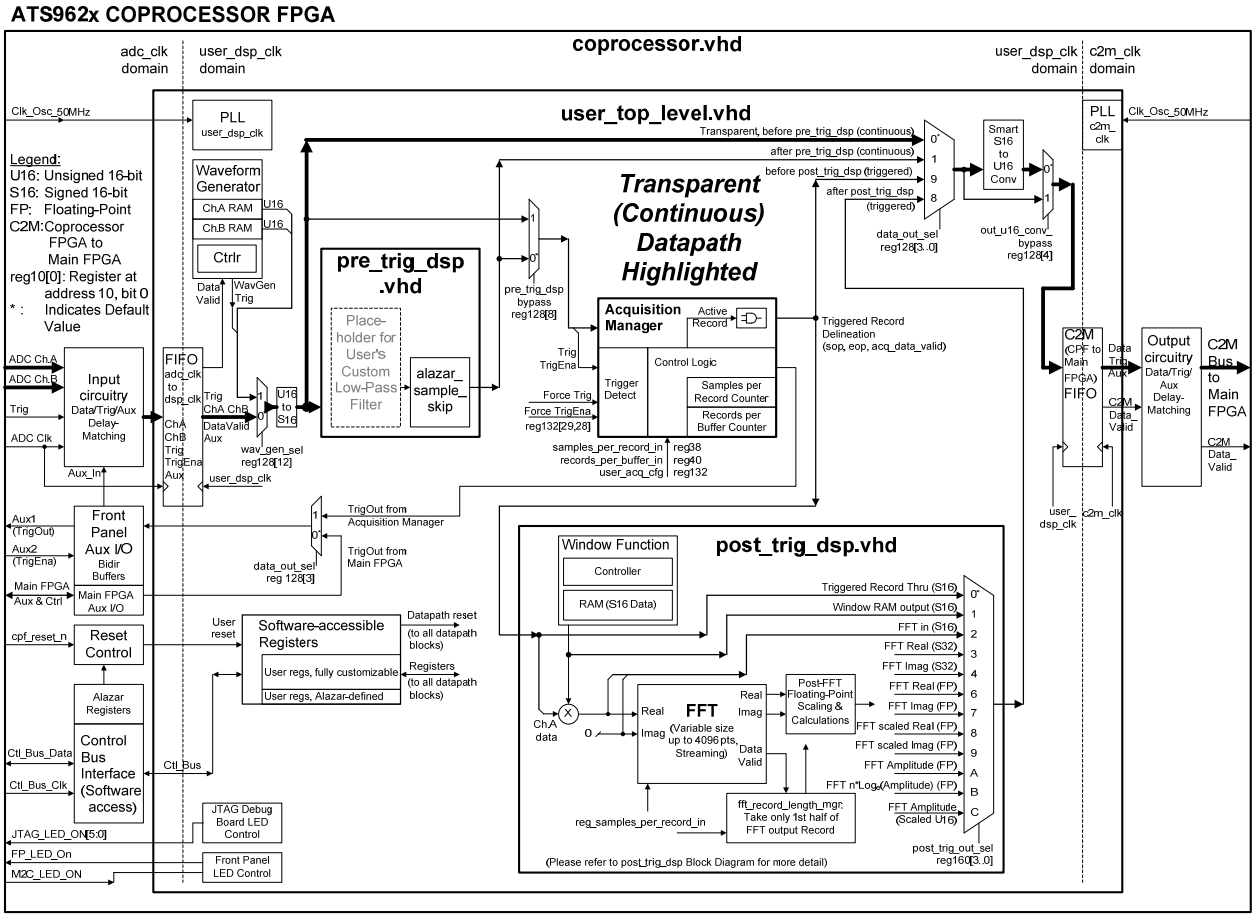


Figure 17-5 Transparent Datapath (Default at Startup)

The following register value is used to obtain this path:

- Register 128: 0x0000_0000

17.6 Auxiliary Ports

The ATS9625 Waveform Digitizer board has 2 Auxiliary Input/Output connector ports on its faceplate.

17.6.1 AUX1 (TrigOut)

AUX1 is a Trig_Out output. When in Continuous operation, it comes directly from the Main FPGA's Trigger engine, and pulses to indicate a trigger event.

When in Triggered operation, it pulses high for approximately 500 micro-seconds everytime the CPF's Acquisition Manager triggers.

17.6.2 AUX2 (Trigger_Enable)

In the ats9625_cpf_multiclk design, the ATS9625 Waveform Digitizer's Front Panel AUX2 port is assigned as an input, and takes on Trigger_Enable functionality. This is an optionally-used signal that goes to the Acquisition Manager, and is an edge-triggered input which needs to occur before Trigger edges are looked for.

© 2003-2012 Alazar Technologies Inc.

18 Detailed Functional Description

This chapter covers each block's functions in detail.

18.1 Datapath Clock Domains

The Multiple Clock Domain framework builds upon the single-clock domain framework, but adds 2 more clock domains in order to provide maximum robustness and flexibility for applications that require it. The clock domains are:

1. **adc_clk**: This is sourced by the Analog-to-Digital converters. Sample data is received from the ADCs along with this clock by the Coprocessor FPGA. Data is taken in from the ADCs and written to the adc-to-dsp FIFO.
2. **user_dsp_clk**: This is the main datapath signal processing clock. Typically all of the user's design runs off this clock. Data is read from the adc-to-dsp FIFO, processed as desired, then written in the c2m FIFO.
   The clock itself is an FPGA internal clock, derived from an independent local 50MHz oscillator, and converted to any desired operating frequency with an internal FPGA PLL. The user can change this operating frequency as needed.
3. **c2m_clk**: This is the Coprocessor FPGA to Main FPGA clock. It is generated by the Coprocessor FPGA provided to the Main FPGA along with the Coprocessor's output data. The Main FPGA is slaved to this clock for its data receive circuitry.

18.2 coprocessor.vhd Block

The coprocessor.vhd block is the FPGA chip's top-level block. In Figure 17-2, it is the outermost block. It takes care of all the I/O interfaces, as well as many of the tedious interconnect functions and low-level protocols.

Alazar has taken care of all these functions in order to facilitate the user experience, while still providing the user with complete design flexibility by placing key customizable functions in the user_top_level Block (e.g. PLLs, software registers, datapath FIFOs, etc.)

The files are located in the alazar_source_files section and should not be modified by the user. The user risks damaging the board by making changes to this block. Please contact Alazar if you believe there is a need for you to change something in this block in order to achieve your desired custom functionality.

The coprocessor.vhd block contains the the following functional blocks: ADC Input, C2M (Coprocessor FPGA to Main FPGA) output, Auxiliary and Trigger I/O and Delay-matching, reset control, and software control bus interface.

18.3 user_top_level.vhd Block

The user_top_level embodies all the key datapath functions and clock domain transfers. Referring to Figure 17-2, it is the large block within the coprocessor.vhd block.

© 2003-2012 Alazar Technologies Inc.

Each major functional block it contains is described in the following sections.

18.4 Waveform Generator

The Waveform Generator is basically a programmable RAM-based sample reader which output selected RAM contents at the same rate as the ADC's encode rate from adc_clk. It is rate-matched to the ADC rate.

There is one Waveform Generator Controller, which simultaneously and synchronously controls 2 Waveform Generator RAMs, one for Channel A and one for Channel B. Each one contains up to 4096 samples, each sample being 16-bit. The RAMs' data format is U16 (unsigned 16-bit integer, binary offset such that the zero value is at 0x8000). Each RAM address location represents one sample.

The Waveform Generator is useful in many instances, namely:

- Standalone use of the Waveform Digitizer board, for software or FPGA development without needing external equipment (which is often expensive or of limited availability)
- "Playback" of a pre-recorded real-world sample sequence
- Comparison and Verification of real-hardware vs. DSP modelling simulation, using exactly the same input sample data
- Debugging an FPGA custom design

The Waveform Generator also has flexible software-configurable options:

Number of Samples: this is the number of samples to output before going back to address 0. Default is 4096, the RAM size. Choosing a smaller number allows the user to generate a sample sequence that repeats itself multiple times throughout a record. Note that each new record always starts at address 0.

Skip Sample Enable: Enables skipped sample mode, whereby a number of RAM addresses are skipped for each new sample.

Skip Samples: This is the number to add to the previous address location for every new sample, resulting in a skip ratio of 1:n. Note that this is the only way of skipping samples in the waveform generator with a guarantee of no sampling jitter.

Example 1:

You have defined a fixed waveform in your RAM, and you want to triple its frequency to quickly test the FFT's response. Set it to read every 3 samples.
Number of samples = 4096, Skip Sample Enable = 1 (on), Skip Samples = 3
Each record would start with the sample data from RAM addresses:
0, 3, 6, 9, 12, 15, 18, 21...4092, 4095, 2, 5, 8, 11...
Note that it is recommended to set your number of samples to be an integral number of cycles of your waveform for continuity when the RAM "wraps around".

Example 2:

You want to repeat a small sample set of 5 samples, multiple times over within each record.

© 2003-2012 Alazar Technologies Inc.

Number of samples = 5, Skipped Sample Enable = 0 (off)
Each record would start with the sample data from RAM addresses:
0, 1, 2, 3, 4, 0, 1, 2 ,3, 4...

Example 3:

You have a waveform 20 cycles long in the RAM, and you want to quadruple its frequency.
Number of samples = 5, Skip Sample Enable = 1 (on), Skip Samples = 4
Each record would start with the sample data from RAM addresses:
0, 4, 8, 12, 16, 0, 4, 8, 12, 16,

18.4.1 Waveform Generator Trigger

The Waveform Generator controller generates a trigger pulse exactly aligned with the first sample at address 0 of the RAMs. If using the Acquisition Manager, it will trigger on exactly the first sample for every one of its records. Every record will therefore start with the Waveform Generator's address 0 contents. All samples in each record therefore repeat consistenly from one record to the next, thus providing completely deterministic and static input records. In other words, every record has the exact same sample sequence, and all records are identical.

18.5 Input Selector (wav_gen_sel Mux)

This input datapath multiplexer basically selects between live ADC data, and data from the Wave Generator. Both Channels are switched together.

The Trigger signal from the front-panel is also replaced by the Waveform Generator's generated Trigger signal. This means the Acquisition Manager will trigger in sync with the waveform generator's output. Trigger_Enable is not affected.

The selection is handled by the cfg_wav_gen_sel bit; please refer to the memory map for details. Default is ADC data.

18.6 Input U16 to S16 Converter

An Unsigned16 to Signed16 converter follows the WavGen selector near the input. This converts the sample data to Signed16 for use by the blocks internally. The format coming from either the ADCs or the Waveform Generator is Unsigned16, binary offset such that the zero is at 0x8000. The converter simply translates that to a signed value, in two's complement format.

18.7 pre_trig_dsp.vhd Block

The pre_trig_dsp block is where continuous sample processing occurs. A number of datapath processing functions could be performed here, such as low-pass-filtering, decimation, interpolation, etc. Any function that operates on a continuous stream of non-trigger-delineated samples should be implemented in this block.

Please refer to section 17.5.2 for examples of continuous processing functions.

© 2003-2012 Alazar Technologies Inc.

The design includes a Sample Skip circuit which reduces the sampling rate.

18.7.1 Sample Skip

The Sample Skip block is a simple sample skipping circuit that simply "drops" samples from the datapath. This effectively reduces the sampling rate by the specified 1:n ratio. By default it is set to 1:1, meaning it does not drop any samples and does not alter the datapath.

The result of this module is to reduce the effective sampling rate. For example, if the ADC clock is at 250MHz and one sets the sample skip to 1:25, the new effective sampling rate is now 10MHz.

This new effective sampling rate has easily understandable consequences which are sometimes forgotten about, so a reminder could be useful. The sample skipper is not a full-featured decimator, which would include a low-pass filter to prevent undesired effects from a reduced sampling rate.

A few words of caution about reducing effective sampling rate:

1. **Aliasing**: The module reduces the effective sampling rate, so any frequencies above the effective Nyquist sampling frequency will create aliasing in the signal. Any signal frequency above 5MHz in the above example will exhibit aliasing. It is the user's responsibility to ensure the input signal is properly low-pass-filtered prior to the sample skip block (e.g. external signal pre-conditioning, or implementing a custom low-pass filter in the pre_trig_dsp block).
2. **Internal RAM-based Waveform Generator ADC-like Jitter**: When used without sample-skipping (i.e. 1:1 ratio), the internal Waveform Generator has the added bonus of being lock-stepped to the sampling clock and thus has zero jitter, producing the exact same sample values from one record to the next. However when using the sample skip module at various ratios other than 1:1, the data effectively does get sampled and you may get sub-sampling-clock jitter, just like an ADC-based acquisition. Note that, independently of the sample skip module, the waveform generator itself has some built-in sample skipping functionality which can produce zero jitter. Please refer to the Waveform Generator section for more detail.

18.8 Acquisition Manager (with Trigger Engine)

The Acquisition Manager is the controller in charge of managing the acquisition, from the time software starts the acquisition until it is ended. It includes a highly configurable Trigger engine which allows many mode of operation.

Acquisition Manager's main functions summary:

- Select windows of input sample data (called records) and identify them as valid to the datapath circuitry.

© 2003-2012 Alazar Technologies Inc.

- Trigger has zero re-arm time, can acquire back-to-back records with no inter-record delay.
- Monitor the acquisition start and end (via the i_m2c_armed signal)
- Monitor Trigger and Trigger_Enable, which are Front Panel inputs
- Optionally ignore Triggers until a Trigger_Enable edge is received
- Delineate records of user-specified size (samples_per_record), starting at each Trigger edge.
- Acquire the specified number of records (records_per_buffer), then wait for the next Trigger-Enable edge.

Here is a typical example, (simple operation, not using the Trigger_Enable input):
The Acquisition Manager would, once armed by the system, monitor the Trigger (Trig In) signal. Up to this point, no data is identified as valid yet. When an edge is detected on the external trigger, it would mark the corresponding ADC input data sample as the 1st sample of a record, and would mark all following samples as valid. It would count up to the configured number of samples per record (say 4096 samples), then indicate the end of the record, and stop marking samples as valid. It would then monitor for the next trigger edge and repeat the process, until it reaches the desired number of records, or the acquisition is ended by the system software.

18.8.1 Triggering Features

The External Trigger (Trig In) input of the ATS962x Waveform Digitizer is normally used as a Trigger to the Acquisition Manager. The edge polarity is configurable, and external trigger can also be configured to be ignored. Software can always manually trigger by toggling a bit.

Please note that there is no other trigger input mechanism implemented at this point (such as level-based trigger detection within the signal, for example). However the code is provided and written such that the user could create such an addition and add it in to circuit.

Normally a Trigger event initiates a record. Once a record is finished, the acquisition manager looks for the next Trigger to generate another record. The trigger engine has zero re-arm time. In other words will take the next trigger edge immediately upon finishing a record. Thus if triggers are timed perfectly, it could create back-to-back records from a continuous stream of samples without missing any. Note that if this is the desired functionality, one could potentially use the continuous streaming or the self-trigger features.

18.8.2 Trigger_Enable Feature

The Trigger_Enable signal is optional. It is turned off by selecting the self_trig_ena bit in the user_acq_cfg register. Please refer to section 17.2 for the location of the Trigger_Enable external signal input.

When self_trig_ena=0, Trigger_Enable functionality is active. Upon start of acquisition, the Acquisition Manager ignores all Trigger edges until it sees a Trigger_Enable edge.

© 2003-2012 Alazar Technologies Inc.

Once a Trigger_Enable edge is received, it then looks for Triggers and will capture one record for each Trigger received, until "records_per_buffer" records are acquired.

Then it will ignore all subsequent Trigger edges until it detects the next Trigger_Enable edge. The cycle repeats itself until the acquisition is ended by software.

It is important to understand that the Trigger_Enable is an edge-based detection, not level-based. The polarity of the edge can be changed in the user_acq_cfg register.

If the bit self_trig_ena=1, then no Trigger_Enable is needed, the Acquisition Manager self-generates a Trigger_Enable internally. It is like this feature is disabled and the Acquisition Manager only cares about Trigger.

18.8.3 State Flow

The acquisition manager's state flow is as follows:

1) Idle:
- Wait for i_m2c_armed to go high

2) Trig_Ena_Wait:
- Wait for a Trigger_Enable edge. Keep going if self_trig_ena is set.
- If i_m2c_armed is low, go back to Idle.

3) Trig_Wait:
- Wait for a Trigger edge. Keep going if self_trig is set.
- If i_m2c_armed is low, go back to Idle.

4) Acquiring_Record:
- Acquire "samples_per_record_in" samples, by raising the internal data_valid for each sample up to that count.
- Increment Record_count.

5) End_of_Record:
- If i_m2c_armed is low, go back to Idle.
- If triggered_streaming, go back to Acquiring_Record.
- If Record_cnt not equal to records_per_buffer_in, go back to Trig_Wait, except if self_trig is set, then go immediately to Acquiring_Record so no sample is missed.
- If Record_cnt equal to records_per_buffer_in, go back to Trig_Ena_Wait, unless self_trig_ena is set, in which case go to Acquiring_Record if we have a trigger edge of self_trig, else go to Trig_Wait.

18.8.4 Functional Timing Diagram - Simple Example

The following functional timing diagram illustrates the Acquisition Manager functionality when used with the configuration parameters specified below.

© 2003-2012 Alazar Technologies Inc.

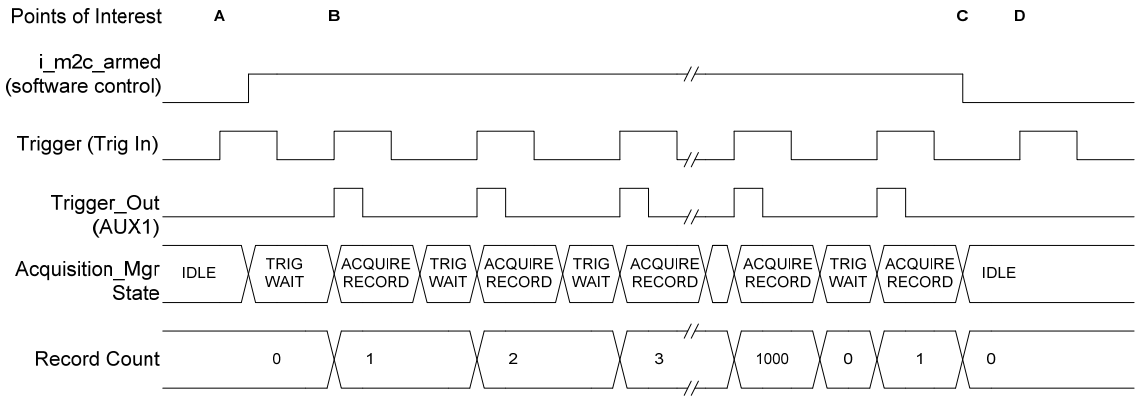


Figure 18-1 Acquisition Manager Functional Timing Diagram (Simple)

Description of Points of Interest in Figure 18-1:

A) i_m2c_armed is low, which indicates that software has not started the acquisition. Trigger is ignored.

B) Software has start the acquisition, i_m2c_armed goes high, Trigger is accepted. Trigger_Out outputs a pulse to indicate that the Acquisition Manager has triggered.

C) Software has aborted the acquisition, i_m2c_armed goes low

D) Trigger is ignored since i_m2c_armed is low.

Configuration parameters for above example:

- "records_per_buffer_in" value of 1000 decimal (0x3E8).
- "user_acq_cfg" value of 0x0001_0000 (self_trig_ena, positive-edge polarities). Setting the self_trig_ena bit makes the Acquisition Manager disregard the Trigger_Enable input.

18.8.5 Functional Timing Diagram - Detailed Example

The following functional timing diagram illustrates the Acquisition Manager functionality in more detail, demonstrating how special cases are handled as well as the use of the Trigger_Enable input.

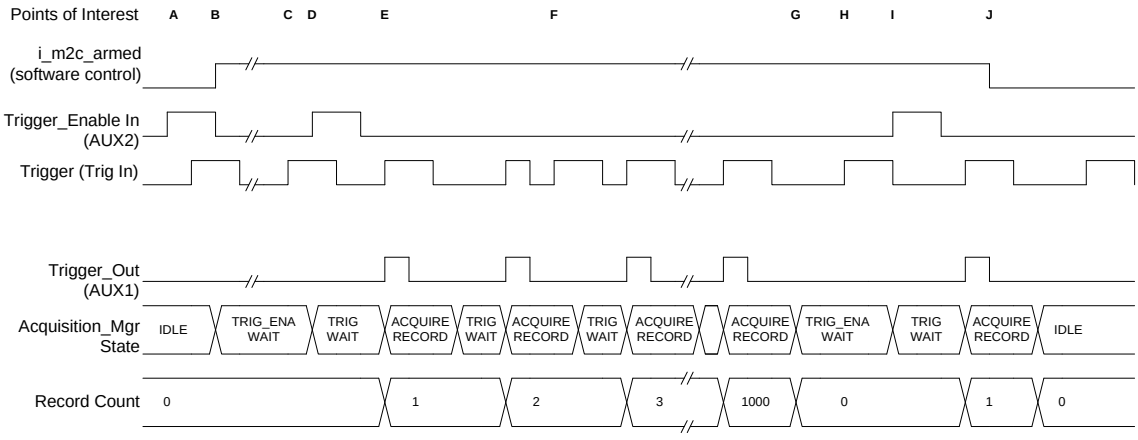


Figure 18-2 Acquisition Manager Functional Timing Diagram (Detailed)

Description of Points of Interest in Figure 18-2:

A) i_m2c_armed is low, therefore all Trigger and Trigger_Enable signals are ignored.
B) i_m2c_armed goes high, Acquisition Manager starts waiting for Trigger_Enable. As Trigger_Ena and Trigger are edge-triggered, their current states when i_m2c_armed goes high are inconsequential.
C) Trigger is ignored because Trigger_Enable has not been received yet.
D) Trigger_Enable edge moves Acquisition Manager into waiting for Trigger edge state. Note that the current Trigger value (high) is inconsequential as it is looking for a trigger edge from this point on.
E) Trigger edge is received. Acquisition of a full record begins.
F) Trigger edge occuring during a record acquisition is ignored, and not "remembered".
G) After reaching 1000 records (which matches the value of our records_per_buffer_in register), this buffer is deemed complete.
H) Same as C. Further Triggers are ignored until we get another Trigger_Enable edge.
I) Same as D.
J) Software aborts the acquisition, causing i_m2c_armed to go low. Here we show this happening in the course of a record, to highlight that the current record will finish being acquired in its entirety per the value in register "samples_per_record_in". This ensures record-based datapath blocks, such as the FFT, don't fall out of record-alignment. Once the record is complete, the Acquisition Manager goes back to Idle state.

Configuration parameters for above example:

- "records_per_buffer_in" value of 1000 decimal (0x3E8).
- "user_acq_cfg" value of 0 (no self-triggers, positive-edge polarities, etc)

18.8.6 Configuration Parameters (Acquisition Manager)

samples_per_record_in: This indicates how many contiguous samples of data get validated by a trigger pulse.

records_per_buffer_in: This indicates how many records should be validated until the next Trigger_Enable is needed. Effectively disregarded if the self_trig_ena bit is set.

self_trig_ena: When this bit is set, the Acquisition Manager will act as if it received a Trigger_Enable edge whenever one is needed, and will thus immediately proceed to look for a Trigger.

force_trig_ena: Allows software to manually force a Trigger_Enable by transition from 0 to 1. Note: Only effective when the Acquisition Manager is in a state where it's looking for a trigger_enable.

trig_ena_ignore: Ignore the Trigger_Enable input signal from the Front Panel. Note that the software force_trig_ena is still operational. This can be used to disregard the external connector and only use a software trigger_enable.

© 2003-2012 Alazar Technologies Inc.

trig_ena_negedge: Detect a negative edge (instead of the default positive edge) on the Trigger Enable input. Does not apply to the force_trig_ena.

self_trig: When this bit is set, the Acquisition Manager will act as if it received a Trigger edge whenever one is needed, and will thus immediately proceed to acquire a record.

force_trig: Allows software to manually force a Trigger. Note: Only effective when the Acquisition Manager is in a state where it's looking for a trigger.

trig_ignore: Ignore the Trig In input signal from the Front Panel. Note that software force_trig is still operational. This can be used to disregard the external connector and only use a software trigger.

trig_negedge: Detect a negative edge (instead of the default positive edge) on the Trigger input. Does not apply to the force_trig.

18.8.7 Triggered Streaming

Triggered Streaming is enabled by the "triggered_streaming" bit in the user_acq_cfg register. Upon start of acquisition, the Acquisition Manager goes through the typical steps of looking for a Trigger_Enable and then a Trigger. However once triggered for the first time, it will acquire back-to-back records continuously without skipping any sample, without further need for any Trigger_Enable or Trigger.

- triggered_streaming=1

18.8.8 Continuous Streaming Mode

Continous Streaming Mode is a mode whereby back-to-back records are acquired continuously without skipping any sample, and the acquisition starts immediately upon software issuing an acquisition start, without need for external trigger signals. The mode is obtained by setting the following bits in the user_acq_cfg register.

- self_trig_ena=1
- self_trig=1
- triggered_streaming=1

18.8.9 Acquire <n> records then stop

The purpose of this acquisition mode is for software to start the acquisition, then the Coprocessor FPGA captures n records (based on Trig In's) then stops. This mode is achieved by using this configuration.

Set this bit only in the acq_cfg register:

- Set user_acq_cfg register: trig_ena_ignore=1
- Set records_per_buffer_in to the number of desired records n.
- Toggle user_acq_cfg register: force_trig_ena bit as described.

Start the acquisition, then toggle the bit "force_trig_ena". At that point the engine will acquire n records and stop. Then, software can decide whether it aborts the acquisition, or issues another force_trig_ena to get another n records. Optionally, setting self_trig

© 2003-2012 Alazar Technologies Inc.

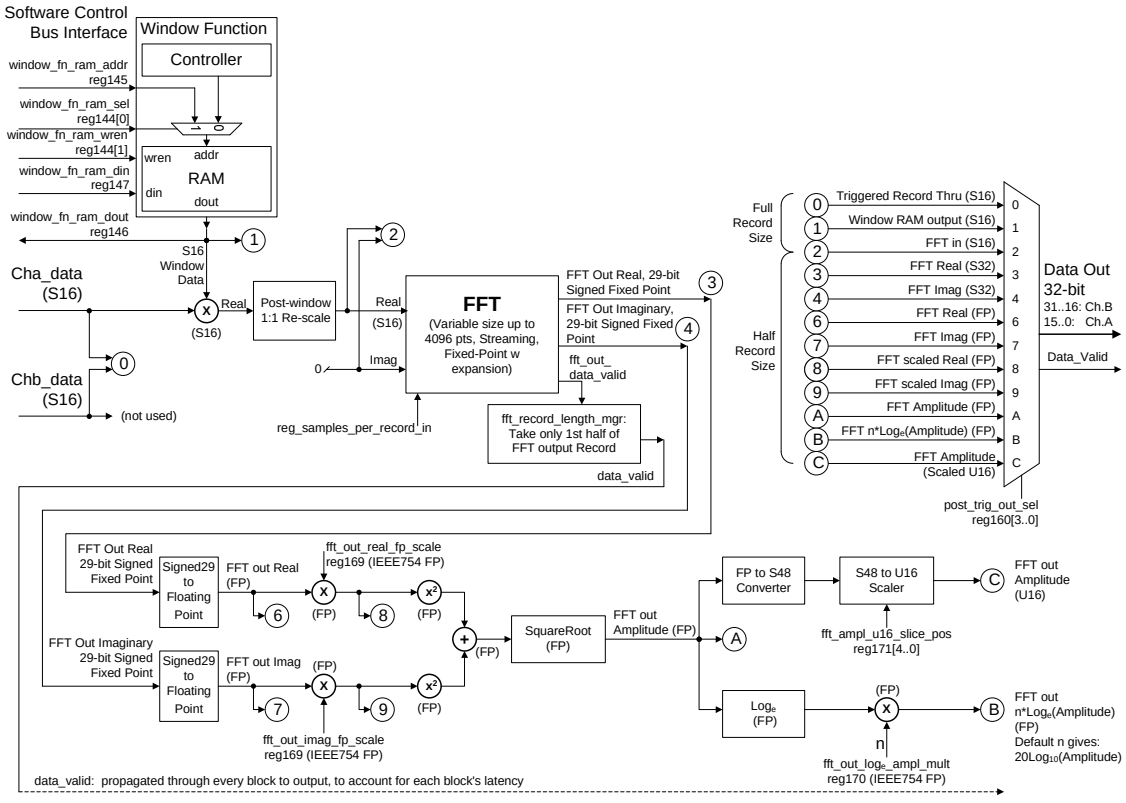
will capture n consecutive records immediately after toggling the force_trig_ena, without needing external trigger input.

Note: If not aborting the acquisition after n records, it is sometimes desirable to acquire additional records such that the n'th record gets pushed through the internal system FIFOs. Please read about Software and System-level considerations in the ATS962x Coprocessor FDK-Designer's Guide, to ensure all of your data comes through the system to the software buffers.

18.9 post_trig_dsp.vhd Block

18.9.1 Block Diagram

Below is a block diagram which illustrates functional blocks and data flow in post_trig_dsp.vhd. Cha_data is the Channel A input data.



U16: Unsigned 16-bit
Sxx: Signed, xx-bits (two's-complement representation)
FP: Floating Point (32-bit IEEE754 single-precision).
Note: register entries for multipliers must be entered in IEEE754 binary representation
Examples: 1.00 = 0x3F80_0000
8.68588964= 0x410A_F967

Figure 18-3 post_trig_dsp.vhd Block Diagram

Each numbered circle represents points in the circuitry where the data can be tapped out and output on the datapath. This is done by selecting the appropriate value for the output selector multiplexer.

18.9.2 Dataflow

Please note that data only flows into the user_datapath block when the Acquisition Manager is Triggered and creating records. That is the only data that flows in, as validated per the acq_data_valid.

Please refer to section 17.4 - Datapath Flow.

If you are not receiving any data on the software side, please ensure that your external equipment setup and software configuration is such that the acquisition manager is triggering.

18.9.3 Window Function

As can be seen in Figure 17-2, a window function is provided in the user_datapath.vhd block, prior to the FFT. The channel A input data is multiplied by the contents of the window function RAM to form a windowed signal as an input to the FFT.

A 4096-point Hann (also known as Hanning) window is preconfigured in the RAM, which the user may reconfigure as desired, either via the FPGA compiler .mif file, or via software write commands.

Once the acquisition manager begins a record, the first input sample is multiplied by the entry at address 0 of the window RAM. The 2nd input sample is multiplied by the entry at address 1 of the window RAM, and so on. Thus if performing a smaller-size FFT, for example a 2048-point FFT, one must pre-fill the window RAM contents with an appropriate 2048-point window data. Otherwise the first 2048 points of the existing 4096-point window will be used, which will certainly not result in a desirable window function.

NOTE: The window function RAM data **must** be updated by the user for any FFT size different than 4096-point, for proper results.

Note that the output selectors allow viewing the window function RAM data as well as the resulting windowed (post-multiplier) data.

The window function data is expected to be of the following format:

- Signed 16-bit

In most cases, the range of the window RAM data will therefore be from zero to the positive maximum, i.e. 0x0000 to 0x7FFF.

© 2003-2012 Alazar Technologies Inc.

18.9.3.1 Post-Window Function Re-Scaler

The window function multiplies the sample with the window data, which of course results in scaled-up values. The post-window function rescaler brings the multiplier result values back to a 1:1 scale with the sample data prior to the window function.

18.9.3.2 Window Function RAM - Viewing and Changing

The are 3 possible methods for viewing and changing the contents of the Window Function RAM.

1. Change the source file compiled with Quartus (ram_window_fn.mif)
2. Use Quartus built-in tool, the In-System Memory Content Editor, along with the USB Blaster or equivalent and the JTAG Debug Adapter.
3. Software read/write accesses.

For method 1, simply edit the .mif file by opening it with Quartus' and using its built-in Memory editor. Save the file, recompile the FPGA, and download the new FPGA to the board.

For method 2, please refer to detailed instructions in the ATS962x Coprocessor FDK-Designer's Guide. There is a chapter devoted to the use of the In-System Memory Content Editor. The Window Function RAM has the Instance ID: "WNDW".

Method 3 involves use of the software interface into the Coprocessor FPGA. Please note that the ATS-SDK supplies example code as read and write to RAM functions, so you do not have to write this from scratch.

The operation is as follows. Please refer to the Memory Map for register locations.

a) Select the RAM for access via the "ram_ctrl" register. Set the appropriate bit to select the Window Function RAM. Note that this disconnects the RAM from the datapath controller.
b) Write the RAM address of interest, in the "window_fn_ram_addr" register.
c) The data content (read data) of that address is immediately available in the "window_fn_ram_dout" register.
d) If writing, write the desired write data in the "window_fn_ram_din" register.
e) Toggle (0-1-0) the appropriate "wren" bit in the "ram_ctrl" register.
f) When done, don't forget to connect the RAM back to the datapath by clearing the appropriate select bit in the "ram_ctrl" register.

Note: do not assert "wren" when the RAM is not selected with "sel". This will overwrite and corrupt the entire RAM contents in many cases. Should this happen by error, the RAM can always be rewritten with proper values.

Also please refer to the Reset section about RAMs: 18.12.1 - RAM contents not restored by Reset.

© 2003-2012 Alazar Technologies Inc.

18.9.4 FFT (Fast-Fourier Transform) Core

The example design implements a Fast-Fourier Transform (FFT) of the Channel A input signal. The FFT is of variable (software or hardware-configured) length to a maximum 4096 points. The FFT core itself is implemented in a radix-2^2, single-delay feedback architecture, using Fixed-Point calculations that allow for natural width growth throughout the internal FFT stages, growing from its 16-bit input to a 29-bit output.

The FFT engine is an Altera IP MegaCore (IP-FFT), which is normally licensed from Altera and bundled with the full Altera Quartus license. At the time of writing, the license was not included in the free Web edition of Quartus.

Please refer to the Altera documentation for this IP core, which is freely available on the Altera website, at:

http://www.altera.com/products/ip/dsp/transforms/m-ham-fft.html

Please note that the brief description on Altera's webpage refers to a Block-Floating Point representation in the FFT. That is not the case in Variable-Streaming mode. Please download and read the full Altera FFT User Guide for the details.

The FFT IP core has many configurable parameters, which are visible by starting the Quartus Megawizard and editing the project's fft.vhd file. Here is a summary of the chosen parameters at the time of writing this documentation:

- Variable-Streaming, Fixed Point architecture
- Input and Output order are both Natural
- Transform Length: 4096 points
- Data Precision: 16 bits
- Twiddle Precision: 16 bits

These parameters are easily changed from within the Megawizard. Please keep in mind however that most parameter changes will also require corresponding VHDL code adjustments, for example to deal with narrower or wider buses, etc.

18.9.4.1 Configuration Parameters (FFT)

samples_per_record_in: This value, which is also used by the Acquisition Manager, indicates the desired number of points to the FFT operation. The current FFT instantiation handles up to 4096 points. This parameter must not be changed when an acquisition is in progress.

18.9.4.2 FFT Internal Scaling

The FFT block scales up the signal in order to keep as much precision as possible within its fixed-point implementation. According to Altera, the scaling is proportional to the FFT transform length (in this case: samples_per_record_in) in the following manner:

Output width = data precision width + 1 + log2(Transform Length)

In this implementation, we therefore have:
Output width = 16 + 1 + log2(samples_per_record_in)

The maximum Output width (also the default) is when samples_per_record_in = 4096:
Output width = 16 + 1 + 12 = 29 bits

As seen in the block diagram (Figure 18-3), the FFT's 29-bit Signed outputs are directly available if desired.

18.9.5 Post-FFT Output Scaling & Calculations

The FFT block's fixed-point outputs are converted to Floating-Point (32-bit IEEE754 single-precision) values, and all subsequent mathematical operations are performed in that format.

Two independent floating-point multipliers are provided to allow scaling to the user's desired range. With these, the user is given the mechanisms to rescale the FFT output to the desired range. This can be used for calibrating to the desired 0dB level if using the Log output.

Calculations are made using Floating-Point hardware blocks, and are available for output. These consist of Amplitude and $n*Log_e$(Amplitude). Note that the Log function is a natural logarithm operation of base e, also commonly represented as ln. The $n*Log_e$ multiplier is provided to allow the user to convert to Log_{10}, or $20Log_{10}$, if desired (the default).

Amplitude is available both as floating-point for full-precision, and Unsigned16 for quick display in AlazarDSO for example. The U16 output has an independent post-scaler as well.

Please refer to the block diagram (Figure 18-3) and Memory map section for more details.

18.9.6 post_trig_out Output Selector

The post_trig_out_sel bits of the post_trig_cfg register control which data to output from the post_trig_dsp block, as shown in the block diagram of Figure 17-2.

Many different outputs are available, using various different data formats. For a description of the outputs, please refer to the memory map section 19.3.8, as well as the block diagram in Figure 18-3.

© 2003-2012 Alazar Technologies Inc.

18.10 Output Selector (data_out_sel Mux)

The datapath output of the Coprocessor FPGA can be selected from various tap points in the design. The output options at the user_top_level are as shown in top-right section of the block diagram in Figure 17-2.

Some outputs are for Continuous operation and others for Triggered operation, as described in section 17.5.

18.11 Output Smart S16 to U16 Converter

This block performs the opposite conversion that was done entering the FPGA. It is "smart" in the sense that it will only perform a conversion if the data format is declared to be Signed16. All other formats will flow through the converter untouched. Data format is tagged appropriately in the datapath circuitry.

The reason for this converter is that the Main FPGA expects U16 data in all Acquisition modes except Coprocessor-Managed.

The conversion, when active, converts two's complement 16-bit Signed values to Unsigned16, binary offset such that the zero is at 0x8000.

The converter can be force-bypassed if desired, as shown in the block diagram. This is usually not necessary.

18.12 Reset Hierarchy

There are 3 levels of reset signals in this design:

1. Hardware reset pin (i_cpf_reset_n)
2. User circuit Reset (user_reset)
3. Datapath blocks reset (datapath_reset)

The hardware reset is accessible through software and resets the entire CPF. Please refer to the ATS962x Coprocessor FDK-Designer's Guide for more details. This is the only reset that holds the PLLs in reset in this design.

user_reset is held active while i_cpf_reset_n is active, and remains active until the PLLs stabilize. user_reset makes all user registers return to their default values. Asserting user_reset also asserts the datapath_reset.

datapath_reset resets all datapath-related circuitry, and is useful in the case where configuration parameters were changed during an acquisition for example (not recommended). Asserting this reset restores all circuitry to a sane state. It does NOT reset the software register values, they retain their configured values. datapath_reset is the lowest in the hierarchy, and gets asserted whenever either other resets get asserted.

© 2003-2012 Alazar Technologies Inc.

18.12.1 RAM contents not restored by Reset

Some internal FPGA RAMs have pre-configured contents, for example the Waveform Generator RAMs and the Window Function RAM.

It is important to note that none of the Reset signals bring back the original contents of the RAMs. In fact the resets do not change the values in the RAMs; they remain what they were prior to the reset. This is a property of the FPGA chip itself and is not a design choice. The FIFOs, on the other hand, get effectively emptied as their pointers get reset.

The only way to restore the RAMs to their default state is to download the FPGA configuration file again.

18.13 Trigger Pulse Width Requirements

In Continous operation, the Main FPGA's trigger engine is used. Thus the trigger pulse requirements are the same as described in the ATS962x Coprocessor FDK-Designer's Guide.

In Triggered operation, the Coprocessor FPGA's Acquisition Manager is used. It can detect right down to a single cycle pulse. For a single-cycle pulse to get properly sampled, one should ensure the trigger lasts at least 3 cycles of the adc_clk. The same minimum duration is needed for assertion/deassertion. There is no duty cycle requirement. The same applies to Trigger_Enable.

© 2003-2012 Alazar Technologies Inc.

19 Memory Map

The Coprocessor FPGA is accessible by the host PC through the Main FPGA. The address space consists of 14 address bits, each addressing a 32-bit register. This provides a total of 16,384 individual 32-bit locations.

The memory map is divided into 3 global sections:

Table 19-1 Memory Map - Global

Addr (dec)	Section Name	Description
0 - 31	**Alazar section (Reserved)**	This is a fixed section which the user must not modify.
32-127	**User section, Alazar-Defined**	This section is driven by circuitry in the user section, but the definition and locations of the registers is defined by Alazar.
128-16383	**User section, Open**	This is open for the user to modify freely.

© 2003-2012 Alazar Technologies Inc.

19.1 Memory Map - Alazar Section (Reserved)

The Alazar section of the memory is a reserved section of 32 locations that the user must not change. Bit definitions or special functions are further detailed in subsections below, per register.

Table 19-2 Memory Map - Alazar Reserved Section

Addr (dec)	Register Name	RW	Description
0	cpf_signature	R	An Alazar-internal unique identifier for the Coprocessor FPGA. Do not change. (1234_5678h)
1	cpf_version	R	The major and minor version of the CPF Alazar framework code release. Do not change.
2	cpf_variant	R	Upper 2 bytes define the variant of the cpf framework used, namely Standard or Advanced. Year timestamp is also included in the lower 2 bytes.
3	compilation_timestamp	R	Date & Time timestamp which gets auto-updated everytime the design is compiled in Quartus. Provides a truly unique identifier (MM:DD:HH:mm)
4	scratchpad	RW	A 32-bit RW test register which has no connection to circuitry other than the scratchpad_inv register.
5	cpf_config	RW	The main CPF Configuration register.
6	cpf_status	R	The main CPF Status register.
7-11	-- Reserved for future use --		
12	scratchpad_inv	R	A read-only register which shows the inverted value of the contents of the scratchpad register.
13-31	-- Reserved for future use --	-	

19.1.1 cpf_version [Addr 1]

Bit	Bit Name	RW	Rst	Description
31-16	<reserved>	R		Reserved.
15-8	cpf_ver_major	R		The 8-bit value of the major version.
7-0	cpf_ver_minor	R		The 8-bit value of the minor version.

The major and minor version of the CPF Alazar framework code release. Do not change. Please use the version register in the User section for your versioning purposes.

19.1.2 cpf_variant [Addr 2]

Bit	Bit Name	RW	Rst	Description
31-16	cpf_variant	R		The 16-bit value of the cpf framework variant. 0000: Standard framework (cpf) 8000: Advanced framework (cpf_adv)
15-0	year	R		The 16-bit value of the build timestamp year version, in a hex-digit to dec-digit format.

For the "year" description and format, refer to the compilation_timestamp Register Description.

Example: 80002012h: Variant 8000 (cpf_adv), Build timestamp year 2012.

© 2003-2012 Alazar Technologies Inc.

19.1.3 compilation_timestamp [Addr 3]

This register is automatically updated when compiling the design in Quartus, and provides a unique identifier for every build that is made. This is a convenient feature to differentiate between multiple debug compiles of the same version, moreover it provides the exact time at which the load was compiled so makes it easy to trace back to source code changes.
The format is as follows: it should be read out of the register and displayed in a 32-bit hex format. Then each byte holds a value that should be looked at as a decimal number. Thus by reading the register in its 32-bit hex value, a human reader can very quickly interpret the compile time. For clarification:

09151005h: September 15th, 10:05 (10:05am)
10281759h: October 28th, 17:59 (5:59pm)
01020001h: January 2nd, 0:01 (12:01am (1 minute past midnight))

Bit	Bit Name	RW	Rst	Description
31-24	month	R		Month (01-12).
23-16	day	R		Day of month (01-31)
15-8	hour	R		Hour (24-hour format) (00-23)
7-0	minute	R		Minute (00-59)

19.1.4 cpf_config [Addr 5]

Bit	Bit Name	RW	Rst	Description
31-1	<reserved>			
0	user_reset	RW	0	Active High signal which gets synchronized to the i_user_adc_clk domain and goes to user_top_level. Should be used to reset user circuitry. Held active during board reset, auto-cleared a few cycles after.

19.1.5 cpf_status [Addr 6]

Bit	Bit Name	RW	Rst	Description
31-2	<reserved>	R	0	
1	pll50_locked	R	0	Active High signal which indicates that the PLL fed by the external 50MHz clock oscillator is locked. Provided by user logic; disregard if you choose to not use this PLL.
0	user_circuit_is_alive	R	0	Active High signal asserted by the user logic to indicate that it is out of reset and running, PLL's locked, and any other prerequisites the user may deem important.

© 2003-2012 Alazar Technologies Inc.

19.2 Memory Map - User Section, Alazar-Defined

The user section, Alazar-Defined registers range from address 32 to 127 decimal. The circuitry for these registers is in the user section, but they are defined by Alazar so software can interpret the correct information depending on the nature of the cpf functionality. The address map is defined in VHDL in the file called "user_mem_map_pkg.vhd".

Table 19-3 Memory Map - User Customizable Section

Addr (dec)	Register Name	RW	Description
32	user_signature	R	A user-defined signature for the FPGA load, which can be used to identify different CPF designs and functionality. Update as needed. Default: 0000_0001h
33	user_version	R	The major and minor version of the User's design block. Update contents as needed.
34	scratchpad	RW	A 32-bit RW test register which has no connection to circuitry other than the scratchpad_inv register.
35	scratchpad_inv	R	A read-only register which shows the inverted value of the contents of the scratchpad register.
36	data_format_cha	R	A read-only register which shows Channel A's data format and width. This describes the data output from the cpf to the main FPGA.
37	data_format_chb	R	A read-only register which shows Channel B's data format and width. This describes the data output from the cpf to the main FPGA.
38	samples_per_record_in	RW	A software configurable register which instructs the CPF FPGA how many samples coming in from the ADCs are acquired per record. Can be useful for trigger-based record acquisitions.
39	samples_per_record_out	R	The CPF instructs the software as to how many samples per record will be output from the CPF to the Main FPGA. This can be the same as samples_per_record_in, but in some advanced applications it is not. It is the responsibility of the user's cpf FPGA circuitry to provide this information (usually based on samples_per_record_in).
40	records_per_buffer_in	RW	A software configurable register which instructs the CPF FPGA how many records coming in from the ADCs are acquired per buffer. Can be useful for trigger-based record acquisitions.
41	records_per_buffer_out	R	The CPF instructs the software as to how many samples per record will be output from the CPF to the Main FPGA. This can be the same as samples_per_record_in, but in some advanced applications it is not. It is the responsibility of the user's cpf FPGA circuitry to provide this information (usually based on records_per_buffer_in).

© 2003-2012 Alazar Technologies Inc.

Addr (dec)	Register Name	RW	Description
42	sample_skip_ratio	RW	Control for sample_skip circuit inside the pre_trig_dsp block. This 24-bit value indicates the desired sample skip ratio interpreted as 1:n. For example: 1: 1:1 ratio. No sample skipping. (Default) 2: 1:2 ratio. This will pass 1 sample, skip 1 sample. 3: 1:3 ratio. Pass 1 sample, skip 2 samples. ...and so on. 16,777,215: maximum value (0xFF FFFF). 0: Invalid value.
43-46	-- Reserved for future definition by Alazar --	-	
47	data_valid_cntr	R	For informational use & debug. 32-bit counter of data-valids produced for the latest acquisition.
48-127	-- Reserved for future definition by Alazar --	-	

19.2.1 user_version [Addr 33]

The VHDL definition of this register is located in user_version_registers_pkg.vhd. Update as needed.

Bit	Bit Name	RW	Rst	Description
31-16	user_ver_major	R		The 16-bit value of the major version.
15-0	user_ver_minor	R		The 16-bit value of the minor version.

19.2.2 data_format_cha/chb [Addr 36, 37]

This register is used by Alazar software to properly configure the Main FPGA's acquisition parameters. It is important that the values of these registers properly reflect the actual data formats used for the acquisition data to be properly transferred.

Bit	Bit Name	RW	Rst	Description
31	Endianness	R	0	0: Little Endian, 1: Big Endian Note: Currently only Little Endian is supported.
30-16	data_format_type	R	00	00h: Unsigned Integer 01h: Signed Integer 02h: Floating Point (per IEEE 754) 03h: Custom
15-0	data_format_width	R	16d	The number of bits on the c2m_data bus used up by each sample. 00h: Channel disabled 10h: 16 bits 20h: 32 bits

Currently Supported Data Formats:

0000_0010h 16-bit Unsigned Integer.
0000_0020h 32-bit Unsigned Integer.
0001_0010h 16-bit Signed Integer.
0001_0020h 32-bit Signed Integer
0002_0020h 32-bit Floating-Point (single-precision IEEE754)

© 2003-2012 Alazar Technologies Inc.

19.3 Memory Map - User Section, Open (Customizable)

The user open section starts at address decimal 128 and goes up to the top of the available addresses (see table). This address map is defined in VHDL in the file called "user_mem_map_pkg.vhd" and can be customized by the user as desired.

Table 19-4 Memory Map - User Customizable Section

Addr (dec)	Register Name	RW	Description
128	user_cfg	RW	Configuration register, which contains bits to configure the user circuit, for example datapath multiplexer selectors.
129	user_status	R	Suggested status register, which contains bits to monitor the status of the user circuit.
130	datapath_reset	RW	Contains 1 active bit which resets the datapath circuit
131	wav_gen_cfg	RW	Wave Generator configuration register
132	user_acq_cfg	RW	Acquisition Configuration register, mainly for the Acquisition Manager
133-143	-- unassigned --		
144	ram_ctrl	RW	Control for accessing internal RAMs via the Control bus interface, allowing software to read/write to internal memories (and disconnecting it from live circuitry in the process).
145	window_fn_ram_addr	RW	Control bus access to window function RAM: enter desired address to read or write.
146	window_fn_ram_dout	R	Control bus access to window function RAM: Shows data read from address specified above, when this ram is selected for access in the ram_ctrl register.
147	window_fn_ram_din	RW	Control bus access to window function RAM: enter desired data to write.
148-159	-- unassigned --	-	
160	post_trig_cfg	RW	Configuration register for use within post_trig_dsp block.
161-167	-- unassigned --		
168	fft_out_real_fp_scale	RW	Floating-point scaling factor (multiplier) for the FFT's Real output, before amplitude calculations. This should be calibrated per your application needs. Format is 32-bit, single-precision floating point value, specified in IEEE754 binary format. Default: 1.0 (0x3F80_0000)
169	fft_out_imag_fp_scale	RW	Floating-point scaling factor (multiplier) for the FFT's Imaginary output, before amplitude calculations. This should be calibrated per your application needs. Format is 32-bit, single-precision floating point value, specified in IEEE754 binary format. Default: 1.0 (0x3F80_0000)

© 2003-2012 Alazar Technologies Inc.

Addr (dec)	Register Name	RW	Description
170	fft_out_loge_ampl_mult	RW	Multiplier for the Log_e(Ampl), in IEEE754 binary single-precision floating-point format. Default is $20/Log_e(10)$=8.685889638065 (0x410af967), to bring the final output result to an effective: $20Log_{10}$(Ampl).
171	fft_ampl_u16_slice_pos	RW	For FFT U16 Amplitude output scaling. A 5-bit value indicating which bit position (counting from 0) the MSb of the U16 slice should start at within the full-scale FFT output. Min is 15, Max is 31, (unprotected from invalid entries). Default is 24 decimal (0x18).
172-16383	-- For user's purposes --	-	For the user to define and use as needed.

19.3.1 user_cfg [Addr 128]

This register is meant as a general-purpose configuration register, at the user_top_level hierarchy level. Please also refer to the FPGA Block Diagram for a graphical view.

Bit	Bit Name	RW	Rst	Description
31-13	-- unassigned --	-	0000	
12	cfg_wav_gen_sel	RW	0	Select RAM-based Waveform generator vs. ADC. 0: ADC data (normal operation) 1: Waveform Generator
8	cfg_pre_trig_dsp_bypass	RW	0	Bypass the pre_trig_dsp block. 0: No bypass (normal operation) 1: Bypass
4	cfg_out_u16_conv_bypass	-	0	Bypass the S16-to-U16 output smart converter. Note that the converter is "smart" in that, when enabled, it only attempts to convert data flagged as S16. Other data is not affected by the converter regardless of bypass mode. 0: No bypass (Normal operation) 1: Bypass
3-0	cfg_data_out_sel	RW	0x8	Data Out (user_top_level block) selector mux value. This also controls the source of TrigOut: Continuous data selects the Main FPGA's TrigOut. Triggered Records selects the CPF Acquisition Manager's Trigger Output. 0: Data from before pre_trig_dsp block (continuous data). CPF Transparent operation. 1: Data from after pre_trig_dsp block (continuous data). CPF pre-processing operation. 8: Data from after post_trig_dsp block (triggered records of data). Normal FFT operation mode 9: Data from before post_trig_dsp block (triggered records of data). To provide debug visibility

© 2003-2012 Alazar Technologies Inc.

19.3.2 user_status [Addr 129]

This register is meant as a general-purpose status register and currently has no bits assigned.

Bit	Bit Name	RW	Rst	Description
31-0	-- unassigned --	-		

19.3.3 datapath_reset [Addr 130]

This register controls the datapath_reset signal. It contains a single bit for this purpose, which is placed at a location to avoid accidental setting.

Bit	Bit Name	RW	Rst	Description
31-17	-- unassigned --	-		
16	datapath_reset	RW	0	datapath_reset signal control. 0: normal operation 1: datapath_reset active.
15-0	-- unassigned --	-		

19.3.4 wav_gen_cfg [Addr 131]

This register configures the Waveform Generators. Please also refer to the FPGA Block Diagram for a graphical view.

Bit	Bit Name	RW	Rst	Description
31	wav_gen_skip_sample_ena	RW	0	Enable skip sample mode. 0: normal operation 1: skip sample mode enabled
27-16	wav_gen_skip_samples	RW	000	When skip sample mode is enabled, the number specified here will be added to RAM address counter on each clock tick. Thus a value of 1 is equivalent to normal operation; a value of 2 skips every other address content, etc.
12-0	wav_gen_num_samples	RW	4096d (1000h)	The number of samples to cycle through before resetting the RAM address to zero. The value in this register is always applicable, regardless of mode, and it counts the actual number of samples that are output (it is not an address-comparison).

Please refer to the Functional Description of the Waveform Generator for more details on functional operation.

19.3.5 user_acq_cfg [Addr 132]

This register is the acquisition configuration register, and mainly controls the Acquisition Manager. Please refer to the Acquisition Manager functional description for operation details. Please also refer to the FPGA Block Diagram for a graphical view.

Bit	Bit Name	RW	Rst	Description
31-30	-- unassigned --	-		

Bit	Bit Name	RW	Rst	Description
29	force_trig	RW	0	Forces a Trigger event upon transitioning from 0 to 1 (regardless of trigger polarity). Use to force a trigger in the acquisition manager via software.
28	force_trig_ena	RW	0	Forces a Trigger_Enable event upon transitioning from 0 to 1 (regardless of trigger_enable polarity). Use to force a trigger_enable in the acquisition manager via software.
27-26	-- unassigned --	-		
25	trig_ena_ignore	RW	0	When set, the Acquisition Manager ignores the Trigger_Enable input on the Front Panel connector. 0: normal operation 1: Ignore Trigger_Enable Front-Panel input
24	trig_ena_negedge	RW	0	Determines the edge polarity detected by the Acquisition Manager for the Trigger_Enable input on the Front Panel connector. 0: Trigger_Enable positive (rising) edge 1: Trigger_Enable negative (falling) edge
23-22	-- unassigned --	-		
21	trig_ignore	RW	0	When set, the Acquisition Manager ignores the Trigger input on the Front Panel connector. 0: normal operation 1: Ignore Trigger Front-Panel input
20	trig_negedge	RW	0	Determines the edge polarity detected by the Acquisition Manager for the Trigger input on the Front Panel connector. 0: Trigger positive (rising) edge 1: Trigger negative (falling) edge
19	-- unassigned --	-		
18	triggered_streaming	RW	0	Turns on Triggered Streaming mode. 0: normal acquisition 1: Triggered Streaming acquisition
17	self_trig	RW	1	Enables self-triggering, which means a trigger will always be detected immediately when the Acquisition Manager starts looking for a Trigger. 0: normal external trigger based acquisition 1: self-triggering enabled NOTE: Self-triggering is **set** by default so that some data comes through automatically even if no external trigger signal is received.
16	self_trig_ena	RW	1	Enables self-trigger_enable, which means a trigger_enable will always be detected immediately when the Acquisition Manager starts looking for a trigger_enable. 0: trigger_enable based acquisition 1: non-trigger_enable based acquisition. NOTE: Self-trigger-enable is **set** by default, as many applications do not use the Trigger_Enable input.
15-0	-- unassigned --	-		

© 2003-2012 Alazar Technologies Inc.

19.3.6 ram_ctrl [Addr 144]

This register allows you to select internal FPGA RAMs for read/write access via software-accessible registers.
Caution: Make sure to select the RAM prior to performing a write, and finish the write prior to unselecting the RAM.

Bit	Bit Name	RW	Rst	Description
31-2	-- unassigned --	-		
1	window_fn_ram_wren	RW	0	Write-enable. When set, RAM gets written into at the presented address and data from window_fn_ram_din. Caution: Make sure you select the ram using window_fn_ram_sel first! Else you will likely wipe your entire RAM as the controller cycles through the addresses. 0: no action. (Default) 1: Write
0	window_fn_ram_sel	RW	0	Selects whether RAM address is connected to the window function controller or to the control bus for Read/Write access. 0: Connected to Window Function Ctrlr (Default) 1: Connected to Control Bus, for R/W access

19.3.7 window_fn_ram_addr/dout/din [Addr 145-147]

These registers are used to access the contents of the Window Function RAM. They are only functional when the window_fn_ram is selected, per the ram_ctrl register.

The RAM's data format is S16 (signed 16-bit integer, two's complement format). It is worth noting that each RAM address location will get matched to one sample. When accessed through the 32-bit software control bus interface, the 16 bits are mapped to the least-significant bits in the dout and din registers, and the upper 16 bits are not used (zero on read).

19.3.8 post_trig_cfg [Addr 160]

This register is meant as a general-purpose configuration register, at the post_trig_dsp hierarchy level. Please also refer to the FPGA Block Diagram for a graphical view.
Output Format types:

- U16: Unsigned Integer, 16-bit.
- S16: Signed Integer, 16-bit, two's complement format
- S29: Signed Integer, 29-bit, two's complement format. The format of the FFT core's raw output.
- S32: Signed Integer, 32-bit (two's complement format)
- FP: Floating-Point, 32-bit, IEEE 754 single-precision

Outputs of less than 32 bits are least-significant justified in the 32-bit transfer word.

© 2003-2012 Alazar Technologies Inc.

*Note: S16 automatically gets converted to U16 on the output of the user_top_level, unless that is bypassed.

Bit	Bit Name	RW	Rst	Description
31-4	-- unassigned --	-		
3-0	post_trig_out_sel	RW	0xC	Post_trig output (post_trig_dsp block) selector mux value. 0: Triggered Record Thru (*S16 ChA and ChB) 1: Window RAM output (*S16 ChA-mapped) 2: FFT in Real(ChA) & Imaginary data (ChB), (*S16, S16) 3: FFT out Real (S32, extended from S29) 4: FFT out Imaginary (S32, extended from S29) 6: FFT out Real (FP, converted from S29) 7: FFT out Imaginary (FP, converted from S29) 8: FFT out scaled Real (FP) 9: FFT out scaled Imaginary (FP) A: FFT out Amplitude (FP) B: FFT out: n*Log_e(Amplitude). (FP). n is specified in fft_out_loge_ampl_mult register. C: FFT out Amplitude (U16, scaled from FP). Binary scaling specified in fft_ampl_u16_slice_pos register.

© 2003-2012 Alazar Technologies Inc.

20 Miscellaneous Considerations

20.1.1 Datapath/Trigger/Aux Latency Matching Considerations

Please refer to same section in the ATS962x Coprocessor FDK-Designer's Guide.

In this project we have taken care of delay-matching data, trigger and Aux (including trigger_enable input) from a Coprocessor FPGA port perspective. In other words, data, trigger and Aux are aligned just as received on the FPGA's I/O, into the adc2dsp data FIFO.

Note: ADC latency is not currently compensated for, i.e. trigger and aux come in to the Coprocessor immediately from the analog buffer whereas analog data goes through the ADC which has a nominal 16-cycle latency. Therefore triggered records have approximately 16 samples of pre-trigger.

20.1.2 FPGA Timing Violations

At the current time, for the FFT example, the Altera FFT IP core does not always meet the 251MHz timing requirements in Quartus 11.1SP2 from compile to compile. We are hopeful that future versions of Quartus and the FFT IP core will resolve this in the future.

There are also a few reset recovery timing violations, which typically only occur in the High-Capacity version of the FPGA (SE260). These are currently deactivated by a multicycle constraint in the .sdc file which is documented as such. Since this is a coming-out-of-reset type of constraint, we have attempted to guard against reset timing startup issues in the design, but do not make any guarantees. The user should address the reset recovery timings as applicable for their use.

The timing analysis is performed by Quartus' TimeQuest tool, over process variations and temperature variations. We have found that the timing violations are minor enough that no problems appear for typical demonstration of functionality.

AlazarTech would like to bring to attention that if the user is planning on using parts of the example in their final design, careful consideration should be given to these timing violations.

Suggestions for the FFT IP core include: performing multiple compile iterations until timings are met (possibly facilitated by using Quartus' Design Space Explorer), using advanced FPGA fitting techniques (Floorplanning, LogicLock), changing the IP core, or reducing the clocking speed. For the reset recovery, a cascaded reset structure could be implemented as suggested in the Incremental design section of the Quartus manual.

© 2003-2012 Alazar Technologies Inc.

21 Software and System Considerations

21.1 Acquisition Transfer Modes

The contents of this chapter can be found in the ATS962x Coprocessor FDK-Designer's Guide. Please refer to it for details.

It is important to use the proper acquisition transfer mode at the system level, for the Coprocessor FPGA's output data to transfer properly. Section 17.5 covers the various modes of operation and acquisition modes for the MultiClk example; please read it carefully.

Note: There are special End-of-Acquisition considerations in Coprocessor-Managed mode, to make sure all desired records are received.
It is strongly recommended that the user read the Software and System Considerations chapter in the ATS962x Coprocessor FDK-Designer's Guide, especially concerning the end of acquisition & buffer transfer considerations.

© 2003-2012 Alazar Technologies Inc.

22 FPGA Designer Section

This section is intended for the FPGA designer that wishes to make custom modifications to this example design.

22.1 Please read the FDK Designer's Guide

The first thing one should do prior to modifying the FPGA code is read the ATS962x Coprocessor FDK-Designer's Guide. It fully describes the Framework on which the MultiClk example was built, and provides very relevant information about this product.

22.2 Functional Simulation

The design can be simulated at both the Complete FPGA level, and the internal FFT core can also be simulated by itself.

22.2.1 Complete FPGA Functional Simulation

The VHDL testbench provided in the framework provides the user with complete FPGA functional simulation, from controlling the ADC samples coming in to the FPGA, to the data coming out of the FPGA.

Some of the provided functions include:

- Sine wave generation, any frequency and amplitude
- RAM-based Arbitrary waveform generation
- text-file based waveform generation
- dump-to-file output samples
- simulator wave output sample viewing: digital values and analog waveforms.

Please refer to the ATS962x Coprocessor FDK-Designer's Guide for more detail on the testbench functionality.

22.2.2 FFT Core Functional Simulation

The FFT core can also be simulated by itself should the user require detailed characterization (for precision and dynamic range for example), or comparison with existing user software FFT algorithms, etc. The Quartus Megawizard can generate simulation models for this FFT, in the following formats:

- VHDL - already included
- Verilog - not pre-generated
- Matlab (.m) model and testbench - already included

The VHDL model is the one used in the VHDL testbench provided in the framework.

The Quartus Megawizard-generated Matlab (.m) model and Matlab testbench files are also provided for convenience, and can be simulated in Matlab with the user providing input data from a text file. The Matlab model is not supported by Alazar, but is provided by Altera as part of the FFT IP Megacore.

© 2003-2012 Alazar Technologies Inc.

22.2.3 Special Note on Functional Simulation Analog Waveforms

Regarding the analog output waveforms in simulation. Please note that the output bus of the CPF, the C2M bus, is clocked at the c2m_clk rate, which is normally faster than the other clocks. The output data will usually have occasional or frequent data_valid=0. This will make the waveform look distorted when it is not.

22.3 FPGA Code Structure

The Coprocessor FPGA is divided into 2 main sections:

- Alazar section
- User section

The Alazar section is a wrapper around the User section, and its function is to take care of the the various chip interfaces and protocols so the user does not have to be concerned with these. It provides the user with a simple interface in which to insert custom functionality.

The User only needs to be concerned starting at user_top_level interface and should not modify the Alazar files, including the top-level file.

The User section is further divided into 3 sections:

- user_top_level
- pre_trig_dsp
- post_trig_dsp

Please refer to the Block Diagrams and the Functional Description Chapter in this document for more detail.

22.4 Data_Valid and Rate Adaptation

With all these different operating frequencies, it is imperative to keep track of valid data samples vs "empty" clock cycles. The ADC always produces a valid sample each clock cycle, so there is no issue in the adc_clk domain. However once we cross over to the user_dsp_clk and then over again to the c2m_clk domain, a rate adaptation mechanism is required.

This is accomplished by using a Data_Valid signal which accompanies data throughout the internal datapath, and out of the Coprocessor FPGA to the Main FPGA. A clock cycle containing a valid sample is "tagged" by setting this Data_Valid signal to a 1 during that cycle. Using this method, any rate lower than the clock rate can be handled by the circuitry. It is the responsibility of each block in the circuit to appropriately modify and propagate the Data_Valid.

© 2003-2012 Alazar Technologies Inc.

Always delay-match the Data_Valid! When processing the data, most operations will take multiple cycles. It is of utmost importance to delay the Data_Valid by same number of clock cycles, otherwise subtle but important data corruption will occur (repeated sample, missed sample etc). Likewise for the other datapath signals, such as sop, eop, trig, aux, etc.

The Data_Valid mechanism provides maximum flexibility, whereby FPGA circuitry can decimate or interpolate or otherwise change the sampling rate, or even create a custom set of values which are not samples but the result of calculations made on samples, for example.

22.4.1 data_valid in post_trig_dsp

The acq_data_valid (i_post_trig_in_acq_data_valid) is the record-based data_valid that is qualified by the acquisition manager, and is the one that should normally be used. The other data_valid signal (i_post_trig_in_data_valid) is a continuously running data_valid, and should not normally be used; it is provided for future expansion. The pre_trig_dsp block is where non-trigger-delineated processing is expected to be.

22.5 Clocks and PLLs

In the MultiClk design, the only clock that should be changed by the user, if desired is the user_dsp_clk.

22.5.1 user_dsp_clk Frequency

The PLL that generates user_dsp_clk can be changed to whatever frequency is appropriate for your needs.

Here are general recommendations:

1. Read the ATS962x Coprocessor FDK-Designer's Guide, Clocking Schemes section as well as the Clocks section.
2. Consider the rates of data flow carefully, versus the depths of the clock-domain crossing FIFOs. Your circuitry must manage the FIFOs so they don't overflow or underrun, and if they do, then must recover approriately.

The default rate of 251MHz on user_dsp_clk is set to be just higher than the ADC's nominal 250MHz, in order to make sure FIFOs don't overrun. (It is also set lower than 260MHz, the c2m_clk rate.) This frequency may make it difficult to meet timing requirements however. In many applications, sampling rates (adc_clk) lower than 250MHz are used. If that is your case, you can consider lowering the user_dsp_clk rate as well.

Please be aware that the board starts up adc_clk at 250MHz by default, so make sure you set the new sample rate first and reset the CPF after so the FIFO levels are managed.

© 2003-2012 Alazar Technologies Inc.

22.5.2 Changing a PLL frequency

Steps required are:
- Edit the appropriate PLL (pll_clk_osc_50mhz_user) in the Quartus Megawizard.
- Change the output frequency to a valid value close to what is desired. Note the multiply and divide factors.
- Don't forget to change the .sdc file!! It must match your chosen PLL frequency otherwise the TimeQuest timing analysis will not be done properly. You will find the appropriate "create_generated_clock" statement near the beginning of the file. Replace the "multiply_by" and "divide_by" numbers appropriately.

22.6 C2M Bus

The Coprocessor FPGA to Main FPGA (C2M) bus is a 16-bit DDR (Double Data Rate) bus which transports effectively 32 bits of data every clock cycle. The DDR aspect is fully taken care of by Alazar source code and the user only needs to think of it as a 32-bit per clock (Single Data Rate - SDR) bus. However, in simulation, one can see the DDR aspect. The testbench includes translator functions that unravel the DDR data so it can be looked at as SDR.

Its clock comes from the user_top_level's pll_clk_osc_50mhz_c2m PLL. It's important to realize that while this clock is the the user_top_level, the c2m bus and the Main FPGA's interface circuit are slaved to this clock. Furthermore data transfer is gated by the data_valid coming out of the user_top_level (o_user_data_valid).

Data is only seen by the Main FPGA on cycles where o_user_data_valid is high.

© 2003-2012 Alazar Technologies Inc.

23 Design Recommendations

Please read and follow the design recommendations made in the ATS962x Coprocessor FDK-Designer's Guide.

© 2003-2012 Alazar Technologies Inc.

24 Simulation Framework

Please refer to the ATS962x Coprocessor FDK-Designer's Guide for a description of the simulation Framework. The framework has been adapted to support the ats9625_cpf_multiclk design.

© 2003-2012 Alazar Technologies Inc.

25 Glossary

ADC	Analog-to-Digital Converter. Converts an analog input signal into a series of digital samples.
ATS-SDK	Alazar Technologies Software Development Kit. The ATS-SDK's provide the user with an API with which to access and control the Alazar Waveform Digitizer boards.
Buffer	A container of user-configurable size which corresponds to the PCIe DMA buffer size. Data is transferred from the Alazar Waveform Digitizer every time a buffer is filled.
CPF	Coprocessor FPGA
DDC	Digital Down-Converter
DDR	Double Data Rate. As opposed to SDR, DDR provides two words of data per clock cycle, one on the clock's rising edge, and one on the clock's falling edge. This allows twice the transfer bandwidth without doubling the clock's physical frequency.
DUT	Device Under Test. A testing term which refer to the device that is currently being tested (e.g. the user's design), as opposed to the circuitry that tests it (testbench etc.).
FDK	FPGA Development Kit
FFT	Fast-Fourier Transform. An efficient algorithm to compute the discrete Fourier transform (DFT), which is a frequency-domain representation of a time-domain function or signal.
FIFO	First-In, First-Out. This is a buffer/queuing structure with which data is read out in the same order as it was written in.
FPGA	Field-Programmable Gate Array. This is the reprogrammable hardware chip that can perform data manipulation at very high speeds.
I	Refers to the In-phase demodulated signal. Also see Q.
JTAG port	(JTAG = Joint Test Action Group.) The JTAG port, in the context of this AlazarTech product, is a serial debug port which provides access to the FPGA through the JTAG Debug board, which also has user-controllable LEDs. The user connects an Altera USB Blaster® to the JTAG Debug board, and connects the JTAG Debug board to the AlazarTech ATS9625 Waveform Digitizer board.
Multiclk	Short for Multiple Clock Domains; refers to a specific FPGA framework which uses multiple clock domains.
NCO	Numerically-Controlled Oscillator. Synthesizes a digital sinusoidal waveform at a user-specified frequency.
OOR	Out-Of-Range. In this context, qualifies the sample data to indicate the input to the ADC exceeded its supported voltage range.
Q	Refers to the Quadrature (90-degree offset) demodulated signal. Also see I.
Record	A "record" in the context of this document is a group of consecutive samples, usually selected through the use of a Trigger signal or

© 2003-2012 Alazar Technologies Inc.

	other mechanism. Can also be thought of as a window of samples. The number of samples in each record (samples per record) is usually user-determined to suit the desired application.
SingleClk	Short for single clock domain. The entire datapath in the Coprocessor FPGA is clocked from the same clock as the ADC's.
SDR	Single Data Rate. This is the standard way of transferring data, where one word of data is transferred every clock cycle, usually changing on the clock's rising edge.

© 2003-2012 Alazar Technologies Inc.